REPORT DOCUMENTATION PAGE

Public reporting burden for this collection of information is estimated to average 1 hour per response, including the time for reviewing instructions, searching existing data sources, gathering and maintaining the data needed, and completing and reviewing the collection of information. Send comments regarding this burden estimate or any other aspect of this collection of information, including suggestions for reducing this burden, to Washington Headquarters Services, Directorate for Information Operations and Reports, 1215 Jefferson Davis Highway, Suite 1204, Arlington, VA 22202-4302, and to the Office of Management and Budget, Paperwork Reduction Project (0704-0188), Washington, DC 20503.

1. AGENCY USE ONLY *(Leave blank)*	2. REPORT DATE 25.Oct.99	3. REPORT TYPE AND DATES COVERED DISSERTATION

4. TITLE AND SUBTITLE
STOCHASTIC ALGORITHMS FOR LEARNING WITH INCOMPLETE DATA:
AN APPLICATION TO BAYESIAN NETWORKS

5. FUNDING NUMBERS

6. AUTHOR(S)
LT COL MYERS JAMES W

7. PERFORMING ORGANIZATION NAME(S) AND ADDRESS(ES)
GEORGE MASON UNIVERSITY

8. PERFORMING ORGANIZATION REPORT NUMBER

9. SPONSORING/MONITORING AGENCY NAME(S) AND ADDRESS(ES)
THE DEPARTMENT OF THE AIR FORCE
AFIT/CIA, BLDG 125
2950 P STREET
WPAFB OH 45433

10. SPONSORING/MONITORING AGENCY REPORT NUMBER

FY99-340

11. SUPPLEMENTARY NOTES

12a. DISTRIBUTION AVAILABILITY STATEMENT
Unlimited distribution
In Accordance With AFI 35-205/AFIT Sup 1

12b. DISTRIBUTION CODE

13. ABSTRACT *(Maximum 200 words)*

19991116 083

14. SUBJECT TERMS

15. NUMBER OF PAGES
191

16. PRICE CODE

17. SECURITY CLASSIFICATION OF REPORT	18. SECURITY CLASSIFICATION OF THIS PAGE	19. SECURITY CLASSIFICATION OF ABSTRACT	20. LIMITATION OF ABSTRACT

Stochastic Algorithms for Learning with Incomplete Data: An application to Bayesian Networks

A dissertation submitted in partial fulfillment of the requirements for the degree of
Doctor of Philosophy at George Mason University

by

James William Myers

BS, Biology, University of Montevallo, 1981
BSEE, Auburn University, 1983
MS, Computer Science, Troy State University, 1991

Director: Kathryn B. Laskey, Associate Professor
Department of Systems Engineering

Spring Semester 1999
George Mason University
Fairfax, Virginia

Stochastic Algorithms for Learning with Incomplete Data:
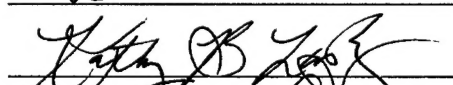
An Application to Bayesian Networks

by

James William Myers
A Dissertation
Submitted to the
Graduate Faculty
of
George Mason University
in Partial Fulfillment of
the Requirements for the Degree
of
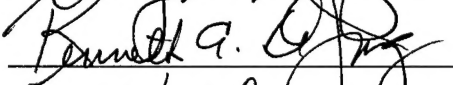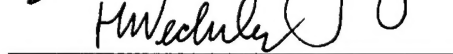Doctor of Philosophy
Information Technology

Committee:

_____ David A. Schum, Dissertation Committee Chair
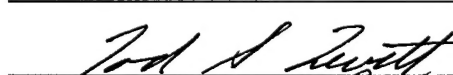
_____ Kathryn B. Laskey, Dissertation Director
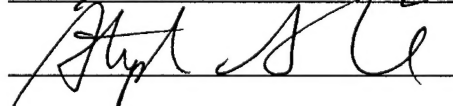
_____ Kenneth A. DeJong
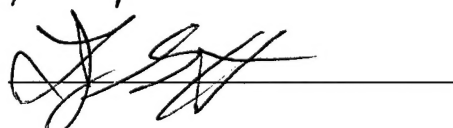
_____ Harry Wechsler

_____ Tod Levitt

_____ Stephen Nash, Associate Dean for
Graduate Studies and Research

_____ Lloyd J. Griffiths, Dean, School of
Information Technology and Engineering

Date: April 14, 1999   Spring 1999
George Mason University
Fairfax, Virginia

DEDICATION

To my mother, Edith

To my Wife, Susie

To my Children; Leslie, Amanda, and Christopher

In loving memory of my father, George W. Myers, Sr.

ACKNOWLEDGEMENTS

Table of Content

List of Figures

List of Tables

ABSTRACT

STOCHASTIC ALGORITHMS FOR LEARNING WITH INCOMPLETE DATA: AN APPLICATION TO BAYESIAN NETWORKS

James W. Myers, Ph.D.

George Mason University, 1999

Dissertation Director: Dr. Kathryn B. Laskey

A goal of machine learning is to develop intelligent systems that improve with experience. To achieve this goal the field has drawn on many diverse disciplines. Because of this diversity, there is no common framework for the field to develop a research vision. This research begins by proposing a machine learning framework. From the framework three hypotheses pertaining to learning Bayesian networks are proposed. The current state-of-the-art learning paradigm for inducing Bayesian networks from incomplete data involves using deterministic greedy hill-climbing algorithms. These algorithms suffer the fate of getting "stuck" at the nearest local maximum. In order to get around this problem, researchers use random restarts. The first hypotheses is that stochastic population-based algorithms will find networks with "good" predictive performance and do not get "stuck" at the nearest local maximum. I demonstrate this hypothesis with an evolutionary algorithm and a Markov Chain Monte Carlo algorithm. A problem with the Markov Chain Monte Carlo algorithm is that it is slow to converge to

the stationary distribution. The second hypothesis developed from the framework is that using global information to propose new states will speed convergence. Finally, because the population-based algorithms have multiple models readily available, I explore the hypothesis that multiple models have a better predictive capability than the single "best" model. I demonstrate these hypotheses empirically with three carefully selected datasets.

Chapter 1 INTRODUCTION

1.1 Learning in Intelligent Systems

One of the main goals of the field of artificial intelligence (AI) is to design computer systems capable of exhibiting intelligent behavior. Winston [Winston 1992] defines artificial intelligence as "the study of computations that make it possible to perceive, reason, and act." Nilsson [Nilsson 1998] adds that AI "is concerned with intelligent behavior in artifacts." These definitions take the rationalist approach to AI. That is, they take the view that we can design intelligent systems on the basis of abstract characteristics of rational, intelligent behavior without necessarily duplicating the exact details of human thought and action. Russell [Russell and Norvig 1995] defines rational action as "acting so as to achieve one's goals, given one's beliefs." The uninteresting solution to designing a system that acts rationally is to pre-program all of its goals and beliefs and let it go about its business. This is not only boring, but fortunately it's not practical except for systems too simple to be called intelligent. Except for toy domains, one can not enumerate or even conceive all the possible situations an intelligent system may encounter in its journey to achieve its goals. Because the universe is large and unpredictable it helps if the agent has the capability of modifying its goals and beliefs given its experiences. This is the process of learning. The field of AI concerned most with this problem has been called machine learning.

A primary goal of machine learning is to build machines that learn from experience (observations from its environment). A machine that learns can be very useful not only for ultimately building an intelligent system, but also for more practical reasons. For example, we may want to build an unmanned submarine that will search the ocean floor for energy sources. It is not possible to give the submarine an enumerated list of all possible situations it may encounter. We may be able to build a model from what little knowledge we have of the ocean floor and hope we capture enough information to allow the submarine to operate autonomously. Perhaps a better solution is an unmanned submarine with a model of the ocean floor and the capability to update the model through its own experience. This example is not a far-fetched scientific dream. It is within our current technological capabilities to build such a learning machine. This is a far cry from the toy problems researchers in machine learning were studying only a few years ago.

Machine learning is a very broad and interdisciplinary field of research that draws on many fields of research outside artificial intelligence and computer science. Among the relevant fields are psychology, philosophy, statistics, Bayesian probability theory, statistical physics, evolution, neural science, cognitive science, computer science, and information theory. Because of this diversity, the field of machine learning has split into many different schools of thought. Researchers from different schools of thought often have difficulty communicating with each other. The rapid influx of new ideas and the diversity of training of scientists in the field make it difficult to establish a common base of discourse.

As early as 1995, the Santa Fe Institute held a workshop that brought together some of the best researchers from different factions of machine learning. The objective of the workshop was to improve communication and cross-fertilization of ideas among the groups. Some results of the workshop were reported by Wolpert [Wolpert 1995]. The conclusions of the workshop were that 1) there is no consensus on a learning framework, 2) there is no consensus on where to go and what issues to answer, and 3) there is no consensus on how to handle assumptions. This was a disappointing outcome for researchers hoping for a meeting of minds. Unfortunately, the situation has not improved significantly since then.

Nevertheless the machine learning field continues making progress. Each of the different groups continues on advances in its own specialty area. There is an unfortunate tendency to overplay the successes of one's own framework and to dismiss criticism from other approaches as uninformed (which it often is).

The field of machine learning is in need of an overarching framework for learning. Such a framework, if it existed, could highlight the common grounds among different approaches, identify opportunities for synergy, identify areas in which research is needed, and in general "point the way" to useful new research agendas. Essentially, a framework that is conducive to scientific inquiry that allows researchers with diverse backgrounds to easily identify where to go and what issues to address.

1.2 A Machine Learning Framework

In developing a machine learning framework one needs to encapsulate the components of any machine learning algorithm. At first glance this task appears daunting

because of the diversity of approaches that draw ideas from many different fields. However, it is possible to identify a few basic components that make up any learning task, and to categorize the approaches taken by different researchers in terms of how they address each of these components

Mitchell, [Mitchell 1997], defines a machine learning system as a computer program that learns from experience given some set of tasks and performance measure, in a way that its measures of performance tend to improve over time. In his Inferential Theory of Learning, Michalski [Michalski 1994] defines this theory of learning in terms of "a goal-guided process of improving the learner's knowledge by exploring the learner's experience and prior knowledge." Kubat, et al., [Kubat, Bratko et al. 1998] describe the learning problem as a system that strives to find a description of a concept given a set of examples and background knowledge. These definitions or variations thereof are the ones researchers with an AI background usually espouse.

The statistician's definition of learning is not much different. Vapnik, [Vapnik 1998] describes a learning model as containing three basic elements: 1) a data generator, 2) a target operator, and 3) a learning machine. The learning machine must choose one of two goals 1) to imitate the target operator or 2) to identify the target operator. Metrics are then used to determine how well the learning machine meets its goals. Though each of the goals is different, they both require the learning machine to produce an appropriate function as output. The appropriate function is found by searching a given set of functions.

Another approach to learning is reinforcement learning [Sutton and Barto 1998]. The idea here is to learn behaviors over time that optimize performance as measured by a numerical reward that depends on the learner's actions. Reinforcement learning starts with an agent that is given a set of explicit goals, an ability to sense its environment, and the ability to choose actions that influence its environment so as to maximize its reward. There are four elements to a reinforcement learning system: 1) a policy which defines how the agent will behave under certain circumstances, 2) a reward function that describes how well the agent is achieving its goal, 3) a value function which specifies how well the agent can expect to do in the long run, and 4) a model of the environment.

From these descriptions and definitions one tends to get a feeling that there are a few basic components common to all learning algorithms. Figure 1.1 depicts the machine learning framework proposed in this dissertation. Each of the boxes in Figure 1.1 can be broken down further in a hierarchical manner.

The next few sections describe the components of the learning framework. I do not claim that the individual models described below are complete, but are structured such that one can easily incorporate new approaches. Following the brief descriptions of the components of the framework, I'll point out a significant enhancement to the overall framework. Following the discussion of this enhancement, I will discuss an object-oriented view of the learning framework that will allow researchers to build more formal models of learning algorithms. Then I will provide some empirical evidence that the learning framework defined herein describes any learning algorithm by sampling and categorizing several papers from machine learning. Next I will demonstrate how

researchers can use the framework to generate hypotheses. Taking the approach described for generating hypotheses I will generate three hypotheses that I implemented and tested. This chapter will conclude with a summary of my research contributions and the structure of this thesis.
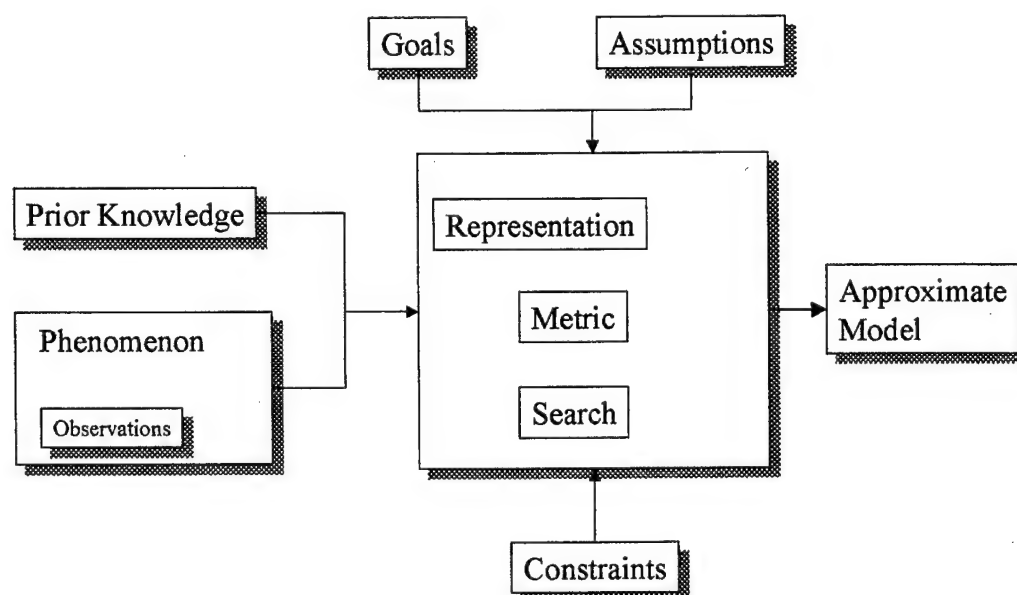
Figure 1-1 Learning Framework

1.2.1 Phenomenon

The crux of the learning problem is to create a model or representation of some phenomenon of nature. The learner must infer the nature of the phenomenon through observable indicators it generates. The observations may be numeric data, images, experiences, or other types of information that can be made available to the learner. In addition, observations may be presented as the tuple $\{X, Y\}$ where X is the input space

and Y is the output space, possibly provided by a teacher. The learning approach that attempts to find the relationship between X and Y is known as supervised learning. When Y is not given explicitly, the learning approach is unsupervised learning. It is possible to use an unsupervised learning approach when Y is given by treating Y as part of the input space. Mathematically, the relationship between phenomenon and observations can be characterized as a function that maps states of the phenomenon to observations. In diagnosis or classification tasks this function is characterized as a mapping from classes or categories to observations from which the class or category is to be inferred (e.g., the function might map diseases to symptoms, or types of equipment malfunction to the results of diagnostic tests). The function generating the observations is unknown to the learner. The objective of learning is to use the observations to infer properties of the phenomenon and/or of the function that relates the phenomenon to the observations. Usually the observations can be decomposed as a set of variables, each of which can take on a set of possible values. An observation is said to be *complete* if it includes values for all the variables; otherwise it is incomplete. The incomplete observation is not necessarily attributable to the phenomenon to be learned, but may be a result of a sensor failure, an occluded view or any other mechanism for the value to be missed. I group incomplete observations here for representational convenience. Observations may also be characterized as deterministic or stochastic. A deterministic model is one in which the values of the observable variables are sufficient to infer with certainty the property the learner is trying to induce. For example, in a classification or concept learning task, observations are deterministic when each observation uniquely

determines the concept or class generating it. Thus for each set of observations S there exists a single class C_1

$$\forall S_1 = s_1, S_2 = s_2, \ldots, S_n = s_n \text{ s.t. } s_1 \land s_2 \land \ldots \land s_n \rightarrow C_1$$

and there is no other class that is consistent with the observation. For the deterministic, complete information case, the problem of learning the model is trivial unless the complete enumeration of cases becomes intractable.

Most observations are uncertain. In other words, there is usually more than one hypothesis about the phenomenon that is consistent with the observations. The most common way to represent uncertainty is to postulate a stochastic relationship between phenomenon and observations: the observations are generated according to a probability distribution that depends on the true state of the phenomenon. A stochastic model may generate observations that have the same values but belong to different classes. Stochastic models are generally more complex to learn than deterministic models with the same number of variables and cases. As in deterministic models the observations can be complete or incomplete.

There are additional classifications possible for observations that I'll only mention in passing. This is not to trivialize their importance but to save time and space. Some learners treat observations as non-deterministic but make no assumptions about whether the process generating them is stochastic, and do not model the associated uncertainty using probability. Observations may be discrete or continuous. They may be ordered in some way such as by time or by value. Observations may be raw sensed data or abstractions. They may be symbols or numbers.

One further point concerning observations and learning is that in order to measure performance one most split the observations into training data and test data (also known as a holdout sample). The training data is the set of observations the learner uses for learning. The test data is the set of observations used to test final performance. The mechanism in which the test and training data are split is by a random draw.

1.2.2 Prior Knowledge

Prior knowledge is what we already know about the phenomenon before attempting to learn from observations. Prior knowledge of the domain is used in all learning approaches. In many approaches prior knowledge is implied in the goals, constraints, assumptions, and knowledge representation. There are learning paradigms that explicitly use prior knowledge as an essential component of the process. The one that is most relevant to this research is the prior probability in Bayesian learning. Bayesian learning is described in some detail in Chapter 2. Prior knowledge is also an important component in Michalski's Inferential Theory of Learning [Michalski 1994], and of most other learning approaches. Prior knowledge may be specified explicitly, or may be embedded implicitly in the representation, algorithms and constraints used by the learner.

When prior knowledge is used explicitly, the learner is given a certain amount of domain knowledge that it uses to help achieve its goals. Explicit prior knowledge can be static or dynamic. Static prior knowledge refers to prior knowledge that is not updated as the system learns from observations. A system with dynamic prior knowledge will update the knowledge to reflect learning.

1.2.3 Constraints

All learning problems are subject to constraints. The constraints may be time, space, syntax, or semantic to name a few. In most learning applications the constraints are implicit in the description of the problem and solution. One learning theory that stands out as dealing explicitly with constraints is the Computational Learning Theory (COLT) [Valiant 1984]. A major goal of COLT is designing efficient learning algorithms in terms of time and space complexity.

1.2.4 Assumptions

Any learning approach applies subject to assumptions about the phenomenon and/or the representation to be learned. For complex or poorly understood phenomena, it is common to make simplifying assumptions to permit the development of a tractable learning approach. One of the more common assumptions is the observations are independent, identically distributed. Another common assumption is the model one chooses to represent the phenomenon, for example a set of rules versus a decision tree. Others include heuristics used during search.

Assumptions are inherently incorrect, but they are necessary to help us build models of the universe. "The universe being a real-number-valued-place, with a truly ludicrous number of degrees of freedom, all of which are quantum mechanical [Wolpert 1995]." One of the questions raised but left unanswered at the workshop at the Santa Fe Institute is whether machine learning researchers should spend more time exploring the why, what, and wherefore of their assumptions. The learning framework allows researchers to explicitly define and describe any assumptions as part of the object model.

1.2.5 Goals

The goals of a learning algorithm define what type of learning it will perform. For instance, many learning algorithms try to learn generalizations of the data. A planning system may attempt to learn an optimal plan. Goals may be implicit in the design of the algorithm or they may be explicitly defined. Michalski defines a learner's goal as specifying "criteria to be satisfied by the output knowledge ... in order to terminate a given act of learning" [Michalski 1994].

Two of the more common goals of generalized models are classification and prediction. For classification, the learner induces a generalized model from a set of input observations and the correct class from a set of classes. After learning, the model is then used to decide on class membership given a new set of observations. In this scenario the classes take on discrete values. This is a form a supervised learning. If the target values are not available to the learner, it has the additional task of learning classes. This is known as clustering. The learner searches for similar input that are close in terms of some distance measure. Another form of classification is regression. The induction in this case is to learn a continuous function from a set of input observations and continuous target values. After the model is built, it is used to make point or interval classifications given a set of input observations.

The learner may also have the goal of inducing models that make predictions. The process of learning a predictor is the same as that of the classifier. However, in many paradigms these models are not the same. A model that predicts takes as input a target class and provides a predicted set of observations. Depending on the representation used

the predicted observations may be given as a deterministic set, such as a set of rules, or as a probability distribution. See [Duda and Hart 1973], [Fukunaga 1990], [Vapnik 1995], [Wolpert 1995], and [Mitchell 1997] for more detailed descriptions of classification and prediction.

1.2.6 Representation

Representation refers to both the type of model the learner uses to represent the phenomenon (its *external model*) and the internal representation used by the algorithm. (Note that the external model is a representation of a phenomenon in the world and the internal representation is a representation of the system's representation). One common classification of internal models is symbolic or nonsymbolic. As more and more complex learning problems are tackled, domain dependent representations are becoming common. An example of a symbolic model is a rule-based system. An example of nonsymbolic models is a neural network. Nonsymbolic models can further be classified as parametric and nonparametric.

The internal representation is implementation dependent. It may be chosen for computational efficiency, representational parsimony, clarity, or other reasons. More than one representation may be used by different parts of a learning system. The internal representation may have the same form of the external model or it may look nothing like the external model. An example of an internal representation that has a different structure from the external model is the binary bit chromosome of a canonical genetic algorithm.

1.2.7 Metrics

Simon [Simon 1986] defines learning as "any change in a system that allows it to perform better...." Mitchell's definition given earlier involved a computer program improving with experience with respect to a performance measure. To evaluate whether a system is learning to perform better requires defining what "better" means. This is the function of the performance metric.

Typically a performance metric is associated with the goals of the learner. For instance the goal may be to find a model that performs well at classifying observations. A standard metric used for such systems is classification accuracy measured as the percentage of observations classified correctly. Another measure of accuracy is some measure of the distance of the system's output from the "true" value, such as the sum of the squared errors. The performance metric is sometimes used on the set of observations used for learning. Many in the machine learning community are recognizing the importance of evaluating performance on cases the system has not yet seen. A common goal in machine learning is a parsimonious approximation. Occam's razor, the famous maxim exhorting scientists to parsimony, states that all things considered equal, the simplest model is preferred. Examples of performance metrics incorporating a preference for simplicity include minimal description length or a count of the number of rules required to cover the positive examples together with a penalty for negative examples covered by the rules. Learning algorithms can, and many times do, use multiple metrics. The metrics used by a learner are domain and goal dependent.

1.2.8 Search

The heart of a learning algorithm is the search through a space of solutions (also known as the landscape) for the "best" solution or set of solutions. Search methods can be categorized as complete or heuristic. Complete search is when all solutions can be easily enumerated so the learner can search over the entire landscape. This problem is not very interesting. A more interesting case of complete search is a problem for which the search space cannot be enumerated, but can be implicitly represented and a tractable algorithm exists to find the optimum. Examples of this include linear regression or iterative proportional fitting for hierarchical loglinear models. However, when the landscape is too large to enumerate or there are time constraints a heuristic search is required. Heuristic search can be classified as deterministic or stochastic. Deterministic algorithms move to the next state in the search space based on a fixed and deterministic rule. Stochastic algorithms move to the next state based on a probabilistic rule. Heuristic search methods can either operate on a single solution at a time or search using a population of proposed solutions. The population based search algorithms can use global information as do genetic algorithms, or they may rely only on local information near each individual solution, as does a population-based simulated annealing algorithm.

1.2.9 Approximate Model

The last component of the learning framework is the learned model, which, except in the simplest of problems, is an approximation to those aspects of the phenomenon of interest to the learner. More formally, the approximate model is a function of search,

metric, and representation given the observations, prior knowledge, goals, assumptions, and constraints. I define the search landscape as

$$M = L(R' \in \mathbf{R} | \mathbf{O}, \mathbf{P}, \mathbf{C}, \mathbf{G}, \mathbf{A}) \tag{1.1}$$

where M is the metric, $\mathbf{R}$ is the representation, $\mathbf{O}$ is the set of observations, $\mathbf{P}$ is the prior knowledge, $\mathbf{C}$ are the constraints, $\mathbf{G}$ are the goals, and $\mathbf{A}$ are the assumptions. Then one can define the approximate model as a mapping

$$F: M, S \rightarrow R^* \tag{1.2}$$

where S is the search mechanism and R^* is the approximate model.

Equation (1.1) indicates that given the observations, prior knowledge, constraints, goals, and assumptions there is a metric that is a function of each potential solution, R'. The function (1.1) is the search space for a solution or set of solutions and can be thought of as a landscape. Equation (1.2) shows the relationship of the metric and the search mechanism. There is a mapping from a search function and metric function to an approximate model, R^*. In other words the search mechanism searches over possible models in the landscape for the maximum (minimum) scoring model. The approximate model, R^*, may not be the global maximum (minimum) model, but represents the "best" model found given the constraints (time and space) of the search.

The learning framework defined above is by no means complete. There may very well be many subcomponents not listed for each of the components of the learning framework. I do claim however, that the top-level components listed in Figure 1.1 accurately model the current machine learning paradigm. There is however, one important improvement one can make to the learning framework.

1.2.10 Discovery Framework

The learning framework described above describes the paradigm used in the vast majority of research and applied work in machine learning. It is a passive model of learning, meaning that the observations are always available to the learner. The learner need only process the observations in accordance with the learning framework and output an approximate model of the phenomenon. This simple framework allows one to learn very complex patterns of observations. Yet it is incomplete.

A more robust learning system includes discovery. Discovery can be thought of as the process of generating new hypotheses and tests for the hypotheses. Discovery is a very special type of learning and as Simon points out [Simon 1986] discovery makes up only a small portion of human learning. Most of the things humans learn they are either told or shown. Arguably though, most of the "leaps" in our understanding of phenomena occur through discovery.

One of the more critical components of a learning system that incorporates discovery is inquiry. Inquiry involves asking questions of the phenomenon to draw out further observations. The new observations allow the learner to refine its hypotheses. The new set of hypotheses may generate additional inquiries that generate more observations and so on. How a learning system can generate inquiries is a subject for further research and is not addressed further in this research. Schum [Schum 1994] provides a good exposition of the processes of discovery and evidential reasoning. Figure 1.2 shows the learning framework with an inquiry feedback loop. I refer to this modified learning framework as the discovery framework.
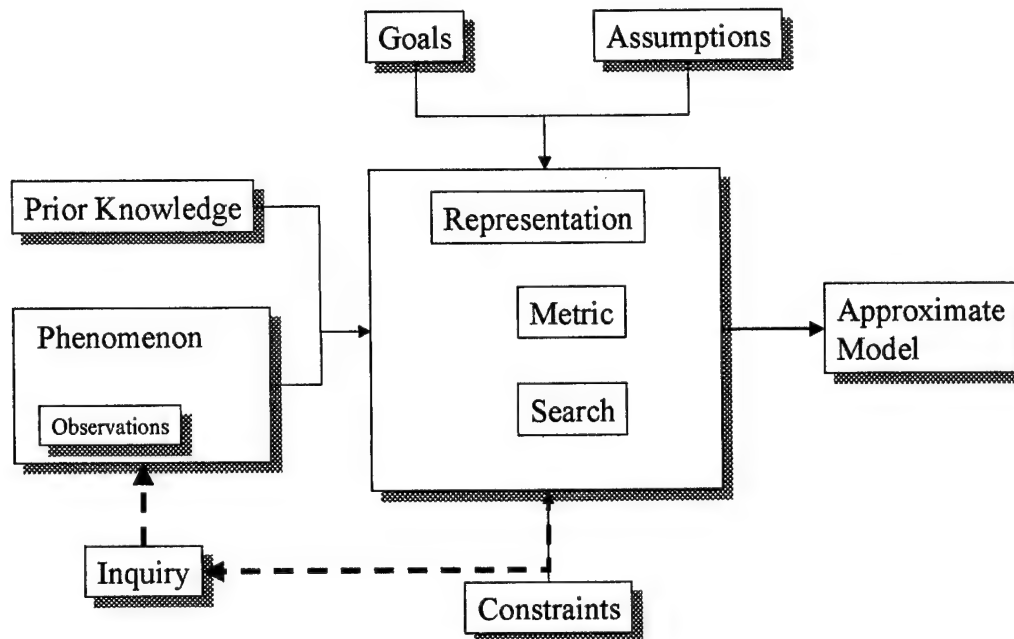
Figure 1-2 Discovery Learning Framework

1.3   An Object Model of the Learning Framework

It is useful to view the learning framework described above as an object model [Rumbaugh, Blaha et al. 1991]. An object model is characterized by four components: classes, objects, polymorphism, and inheritance.   An object model can be used to represent things in a domain and their interrelationships to each other.   The framework in Figure 1.1 can be viewed as an object model.   To specify this object model, each component of Figure 1.1 is represented as a class with associated attributes and methods. Each of these classes can be instantiated to form an object.   Thus, a learning system within the framework of Figure 1.1 can be defined by creating instances of the classes associated with the different components.

In an object-oriented system, the same operation in one object need not behave the same as in another object. This is called polymorphism. This means, for example, that different learning system classes can be defined with very different representations or search methods. Classes can be structured hierarchically so as to allow classes in a hierarchy to share attributes and operations. For example, **search** is represented as a superclass with a hierarchical structure. Its subclasses may be **deterministic** search and **stochastic** search. Each of these subclasses may share attributes and operations from the superclass **search**. Figure 1.3 gives the top-level object model of the learning framework.

Defining the learning framework as an object model has important advantages. First, it is very difficult to enumerate all the different subclasses belonging to the superclasses of the learning framework shown in Figure 1.3. Because the components in Figure 1.3 are superclasses with a hierarchical structure it is trivial for a researcher with a new approach to specify the approach as a subclass in the appropriate hierarchy. In this way, the new subclass inherits all of the attributes and operations from its parent. Further a specific instantiation of the class (an object) in a learning algorithm need not (and probably will not) behave the same as other classes within the same hierarchy. An object model approach describes a framework that provides researchers the efficiency of reusing classes, attributes, and operations while maintaining the needed flexibility of adding new classes to the appropriate hierarchy.

Another important characteristic of the object model of the learning framework is as follows. One can define a hypothetical meta-level learning algorithm that takes as input

the complete (or incomplete) set of classes in the learning framework and outputs learning algorithms. The hypothetical learning algorithm would have the same structure as Figure 1.3. The observations are the set of classes. The prior knowledge, assumptions, and constraints, of course, would have to be provided. The representation is the learning framework. The search algorithm would search through combinations of classes, in the form of complete learning algorithms, using an appropriate metric and set of goals. Each solution (set of classes) the algorithm generates in its search is a hypothesis. Its performance metric is evidence in support of or against the hypothesis. This meta-learning algorithm described above is in fact a hypothesis-generating machine and its implementation is well within current capabilities. Nevertheless, this example demonstrates abstractly how any researcher in machine learning can use the learning framework object model to generate any number of learning algorithm hypotheses. My conjecture is that many researchers generate hypotheses using a mental model that is similar to this learning framework. Nevertheless, to my knowledge the framework described here has not previously been explicitly articulated. Figures 1.4 through 1.11 present the individual object models for each component of the learning framework.

1.4   *Survey of machine learning approaches*

To demonstrate the feasibility and ease of use of this approach I surveyed 30 articles of machine learning techniques and classified them according to the learning framework. The results are provided in Table 1.1.
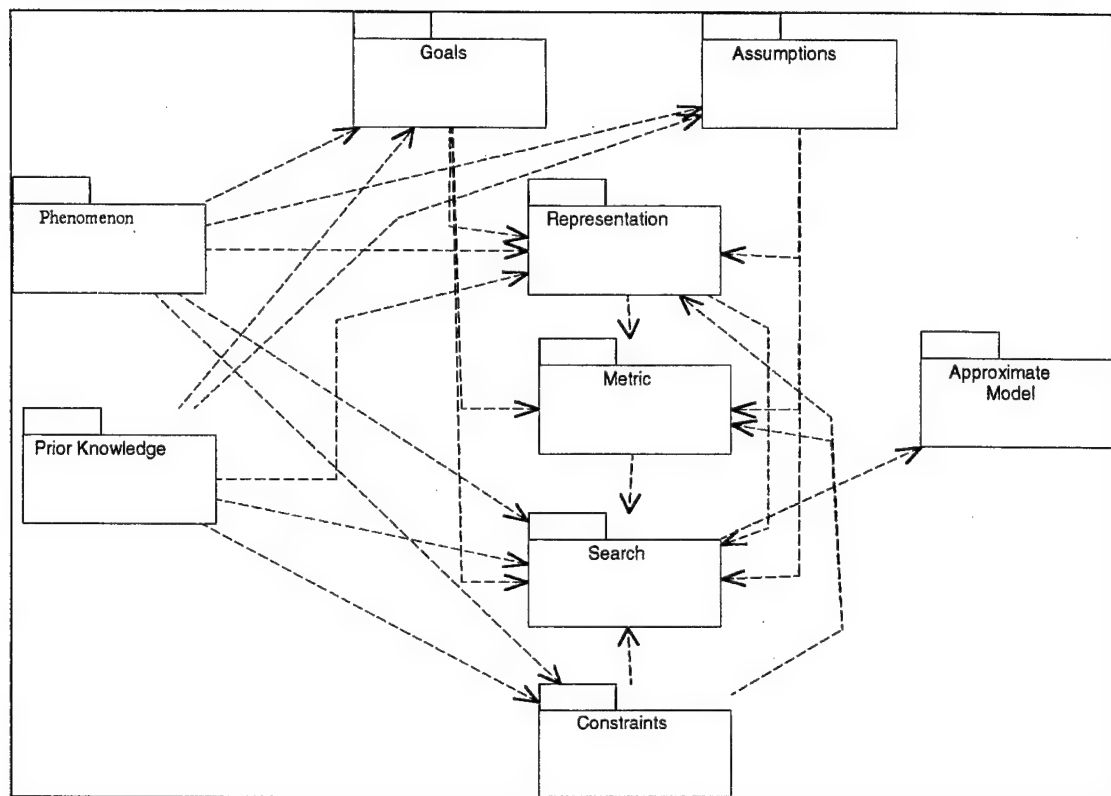
Figure 1-3 Top Level Object Model of Learning Framework

1.5 Generating Hypotheses from the Learning Framework

Now that we have the framework and tabulation we want to use it to identify scientific hypotheses about learning algorithms. The first step is to identify high causal attribution of success (high scoring algorithms) to features of the problem and/or

Table 1-1 Survey of Machine Learning Papers

Paper	P	PK	G	A	C	R	M	S
[Quinlan 1986]	dc,sc, sm	imp	g	ms,h	spc,t	DT	a	ds
[Ginsberg, Weiss et al. 1988]	dc,sc	imp	g	ms,h	spc,t	RB, int-tbl	a	ds
[Gennari, Langley et al. 1989]	dc,sc	imp	g	ms,h	spc,t	RB	a	ds
[Mitchell, Utgoff et al. 1993]	dc,sc	exp	g,pr	ms,h,	spc,t, syn	pr,tree	a	ds
[Mooney 1993]	dc,sc	exp	g,be	ms,h	spc,t	RB	a	ds
[Abe, Li et al. 1995]	dc,sc	imp	g	ms,h	spc,t,syn	RB	a	ds
[Almueallim, Akiba et al. 1995]	dc,sc	imp	g	ms,h	spc,t	DT	a	ds
[Auer, Holte et al. 1995]	dc,sc	imp	g	ms,h	spc,t	DT	a	ds
[Baluja and Caruana 1995]	dc,sc	imp	g	ms,h	spc,t	par	a,spd	dpg
[Cichosz and Mulawka 1995]	dc,sc	imp	pln	ms,h	spc,t	symb	ga	ds
[Weiss 1995]	sc	imp	g	ms,h	spc,t	RB	a	ds
[Brodie and DeJong 1998]	sc	exp	g	ms,h	spc,t	par	a	dc
[Beveridge 1998]	sc	imp	g	ms,h	spc,t	par	a,spd	spg
[Grove and Schuurmans 1998]	sc	imp	g	ms,h	spc,t	nonpar	a	dpl
[Nagata and Kobayashi 1997]	dc	imp	bs	ms,h	spc,t	dom,int_dom	dist	spg
[Merz and Freisleben 1997]	dc	imp	bs	ms,h	spc,t	dom,int_dom	dist	dspg
[Deb and Goyal 1997]	sc	imp	bs	ms,h	spc,t	dom,int_dom	dom	spg
[Whitley, Beveridge et al. 1997]	sc,sm	imp	bs	ms,h	spc,t	dom,int_dom	dist	spg
[Schoenauer and Michalewicz 1997]	sc	imp	bs	ms,h	spc,t	dom,int_dom	dist	spg
[Sarma and DeJong 1997]	sc	imp	bs	ms,h	spc,t	dom,int_dom	dist	spg
[Friedman 1998]	sm	imp	g	dist,ms,mp,h	spc,t	BN	pred	ds
[Geiger, Heckerman et al. 1996]	sm	imp	g	dist,ms,mp,h	spc,t	BN	pred	ds
[Binder, Koller et al. 1997]	sm	exp	g	dist,ms,mp,h	spc,t	BN	pred	ds
[Chickering 1996]	sc	exp	g	dist,ms,mp,h	spc,t	BN	pred	ds

[Cooper and Herskovits 1992]	sc	imp	g	dist,ms, mp,h	spc,t	BN	a	ds
[Jenzarli 1996]	sc	imp	g	dist,ms, mp,h	spc,t	BN	pred	ds
[Larranaga, Murga et al. 1996]	sc	imp	g	dist,ms, mp,h	spc,t	BN	a	spg
[Singh 1997]	sc,sm	imp	g	dist,ms, mp,h	spc,t	BN	a	ds
[Kearns, Mansour et al. 1999]	sc,sm	imp	g	dist,mp, h	spc,t	parm	dist	ds
[Saul and Jordan 1999]	sc	imp	g	dist,ms, mp,h	spc,t	NN	dist	ds

P: phenomenon   d: deterministic   s:stochastic
                       c: complete   m: missing

PK: prior information   imp: implicit
                           exp: explicit

G: goal   g: generalization   bs: best solution
               pln: plan   pr: program
               be: best explanation

A: assumption   dist: distribution   mp: model parameter
                     ms: model selection   h: heuristic

C: constraints   spc: space   syn: syntax
                   t: time   sem: semantics

R: representation   symb: symbolic   BN: Bayesian network
                   par: parametric   DT: decision tree
                   nonpar: nonparametric   RB: rule-based
                   NN: neural network

M: metric   a: accuracy   pred: prediction
               par: parsimony   spd: speed
               ga: goals achieved   dist: distance
               dom: domain dependent

S: search   d: deterministic   s: stochastic
               s: single   p: population
               l: local   g: global

algorithm. In other words, identify components of the learning framework that appear to perform well with other combinations of the framework. It is also important to identify high causal attribution of failure to features of the problem and/or algorithm. The process of finding successful and unsuccessful features of the learning algorithm for a problem is through a detailed literature search. Identify the stated reasons for success and/or failure. The next step is to identify "holes" in the algorithms that have not been explored. Matching the "holes" to components that perform well under similar circumstances will naturally lead to a new set of hypotheses and tests of the hypotheses.

1.6 Three Hypotheses from the Learning Framework

Using the approach identified and Section 1.5 and the learning framework, I identified three hypotheses for the problem of learning Bayesian networks from incomplete data. See Chapter 2 for a detailed discussion. The problem is characterized by a multimodal landscape. Current approaches use deterministic hill climbing search techniques for searching of this multimodal landscape. These search methods get stuck at local maxima and require multiple restarts.

1.6.1 Stochastic Search Hypothesis

One set of search methods that have not been explored to any depth for inducing Bayesian networks from incomplete data is stochastic approaches based on a population of solutions. Evolutionary algorithms and multiple chain Markov Chain Monte Carlo algorithms are both instantiations of this class. These algorithms depend on stochastic acceptance to allow solutions to escape from local maxima. It seems reasonable to assert

that the family of stochastic algorithms based on populations of individuals will perform well on the multimodal landscape of Bayesian network representations and stochastic missing observations.

1.6.2 Global Information Hypothesis

One of the main problems with the Markov Chain Monte Carlo (MCMC) approach is they converge slowly to the stationary distribution. Refer to Chapter 3 for discussion. An active area of research in MCMC algorithms is approaches for speeding up convergence. One of the main reasons to exchange information (crossover) in evolutionary algorithms is to generate potentially better fit solutions. Markov Chain Monte Carlo algorithms do not generally take advantage of information exchange between chains. Information exchange is a means of generating new proposed states (offspring) from global, or population, information. The second hypothesis of this thesis is that use of global information will speed convergence in MCMC algorithms. See Chapter 4.

1.6.3 Single Model versus Multiple Models

Most learning algorithms produce as output a single "best" model. There may remain a great deal of uncertainty about whether this "best" model is correct (in high-dimensional problems, the "best" model typically has tiny posterior probability), yet it is sometimes treated as ground truth. Madigan et al., [Madigan, Raftery et al. 1994] present theory and empirical results suggesting that averaging over multiple models results in better predictive performance than using a single "best" model. Since the stochastic

population search methods explicitly explore using multiple models, this is a good opportunity to explore further the hypothesis that an average of multiple models is a better predictor than the single best model.

1.7 Summary

This thesis extends the field of machine learning in several ways. First, I have developed a learning framework that, if accepted, allows researchers in machine learning to explicate where we have been, where we are, and more importantly where we should be heading. The framework allows researchers to identify problem areas and generate hypotheses. Combined with inquiry it is a framework for discovery learning. Using the learning framework I have identified and articulated three scientific hypotheses. These hypotheses are all associated with a currently active research problem, learning Bayesian networks from incomplete data. Using this problem area I not only have a test case for my hypotheses, but I also achieve an improvement in the state-of-the-art in Bayesian network learning.

The remainder of this thesis is structured as follows. Chapter 2 describes the problem of learning Bayesian networks from incomplete data. It starts with a discussion of probability theory, with an emphasis on the Bayesian approach. Then it briefly describes graphical models and Bayesian networks which are specific instances of graphical models. Finally, Chapter 2 describes the current state-of-the-art in learning Bayesian networks. Chapter 3 describes in some detail the stochastic population based algorithms, the evolutionary algorithms and the Markov Chain Monte Carlo algorithms. Chapter 4 develops a new algorithm, Evolutionary Markov Chain Monte Carlo, that takes

advantage of the benefits of both the evolutionary algorithm's global information exchange and the Markov Chain Monte Carlo algorithm's basis on the first principles of Probability theory. In Chapter 5 I'll describe the empirical approach taken and give the results of some experiments conducted to help select a "good" set of algorithm parameters. Chapter 6 details the results of comparative experiments for each of these algorithms. Finally Chapter 7 summarizes the thesis and provides ideas for further research.
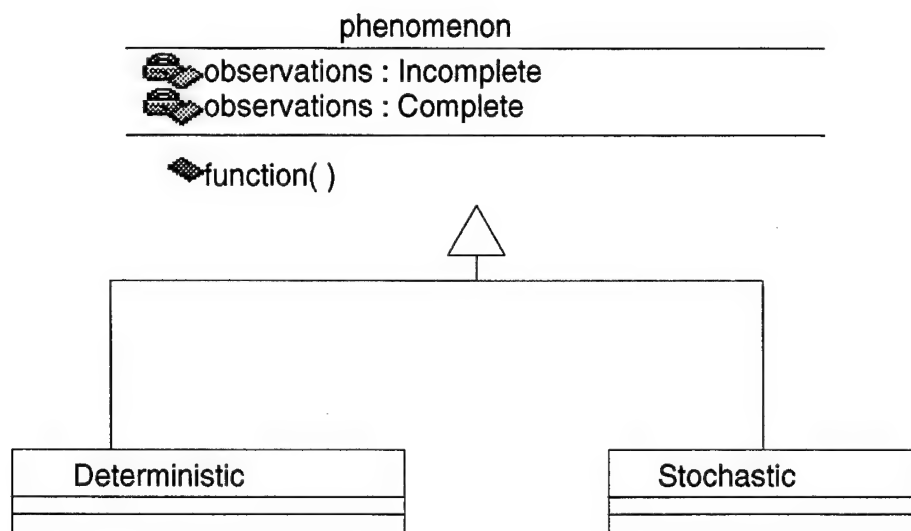
Figure 1-4 Object Model for Phenomenon
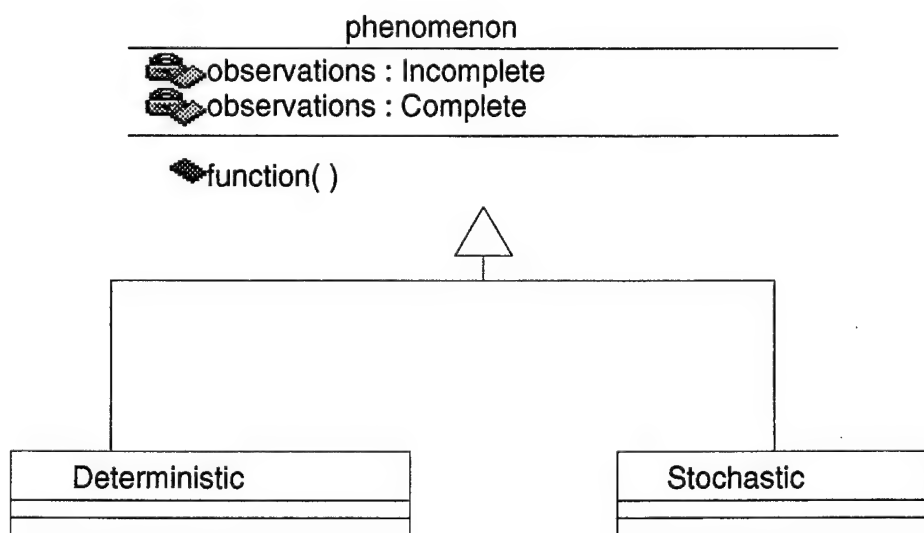
Figure 1-5 Object Model for Prior Knowledge
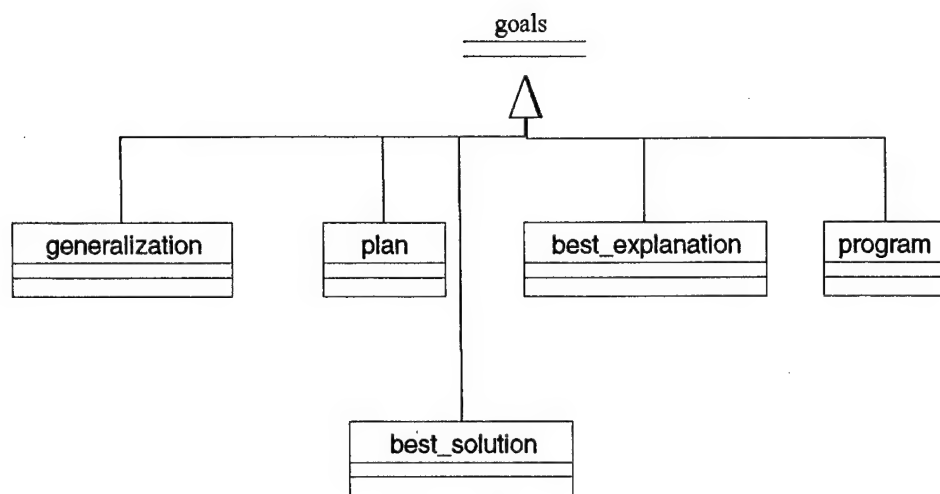
Figure 1-6 Object Model for Goals
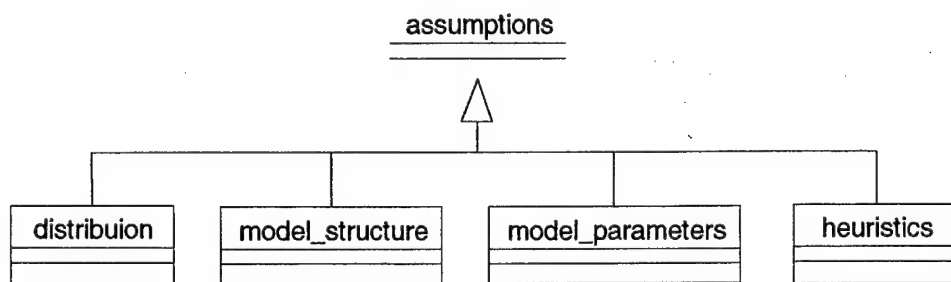
Figure 1-7 Object Model for Assumptions
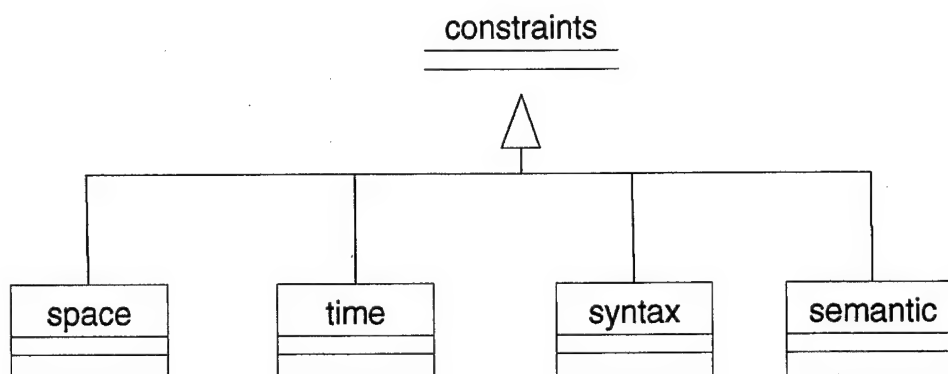
Figure 1-8 Object Model for Constraints
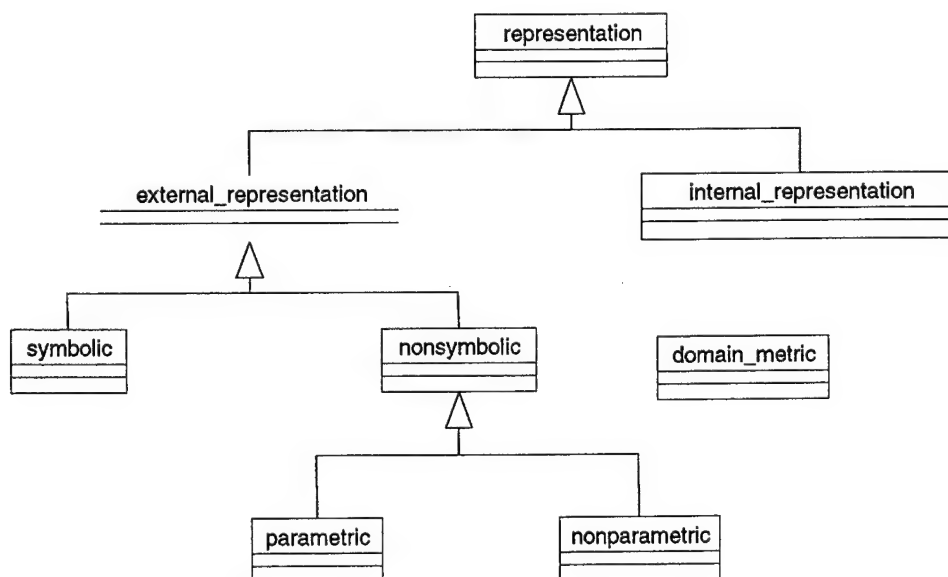
Figure 1-9 Object Model for Representation
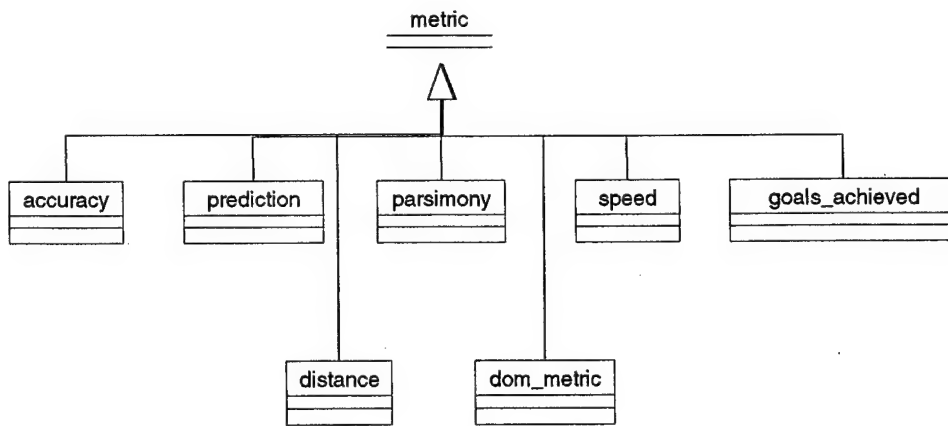
Figure 1-10 Object Model for Metrics
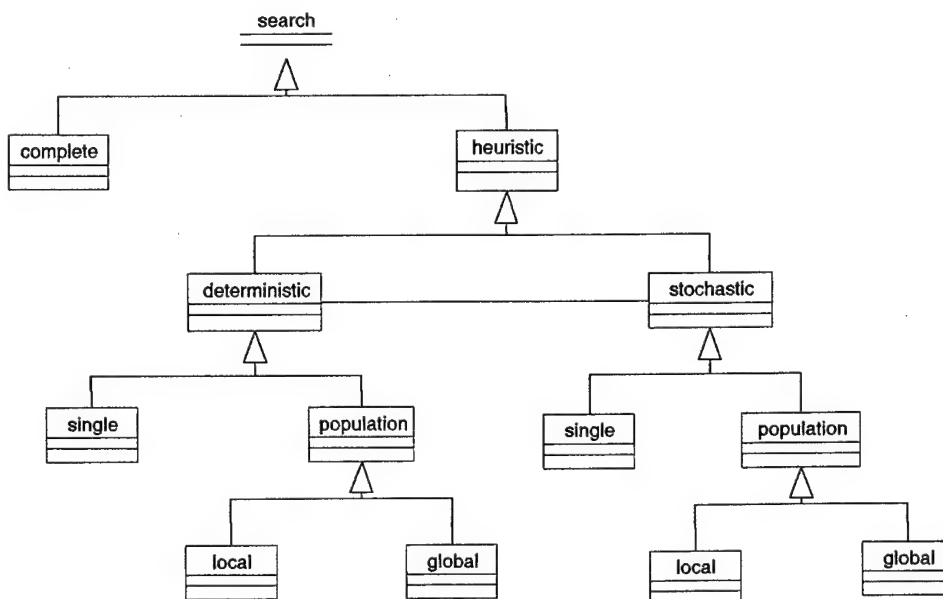
Figure 1-11 Object Model for Search

Chapter 2 LEARNING BAYESIAN NETWORKS

This chapter provides the necessary background information for understanding the problem of learning Bayesian networks from incomplete data. It begins with an introduction to probability with an emphasis on the Bayesian paradigm. This leads nicely to a brief discussion of graphical models and describes Bayesian networks as a specific type of graphical models. This chapter ends with an introduction to learning Bayesian networks from data and a survey of the state-of-the-art approaches to learning from incomplete data.

From the perspective of the learning framework this chapter addresses the phenomena, prior knowledge, goals, representation, and many assumptions and constraints. It also addresses previous work on this problem with emphasis on the search component.

2.1 Probability

Until recently, probability theory was considered as being unimportant in the search for an intelligent system. Perhaps this stems from the early days of computing when AI researchers realized that computers could be used for something other than processing mathematical formulas faster and more accurately than humans. Many attribute the birth of AI to Turing. In his seminal paper, "Computing Machinery and Intelligence" he described what is known as the Turing Machine and discussed the

proposition "Can machines think?" [Turing 1995]. From Turing's first paper, AI was born. Early approaches for developing intelligent machines focused mainly on symbolic logic. The key to discovering intelligence was thought to lie in this formalism. In fact, Newell and Simon stated that "symbols lie at the root of intelligent action, which is, of course, the primary topic of artificial intelligence." [Newell and Simon 1995] Probabilistic approaches were not looked upon with great favor even though some of the greatest scientists in history considered probability an important aspect of intelligence. Take for example a quote by J. Clerk Maxwell given in Sir Harold Jeffreys' seminal book on probability theory [Jeffreys 1998]

> They say that Understanding ought to work by the rules of right reason. These rules are, or ought to be, contained in Logic; but the actual science of logic is conversant at present only with things either certain, impossible, or entirely doubtful, none of which (fortunately) we have to reason on. Therefore the true logic for this world is the calculus of Probabilities, which takes account of the magnitude of the probability which is, or ought to be, in a reasonable man's mind.

There are probably many reasons for the reticent use of probability theory in early AI research. A couple of significant roadblocks come to mind. One potential reason may be due to the most prevalent view of probability theory; the frequentist approach. Although the early concept of probability was that it was an extension of logic, that view is not currently the most prominent. The most popular view of probability, and in fact the one taught in most probability classes, is that it is a measurable regularity of a phenomenon characterized by repeated random trials [Feller 1968]. This approach does not put merit on the probability of a single event. An AI researcher with frequentist probability training would not even consider this as relevant to the problem at hand.

Many inference tasks for an intelligent system involves single events, such as what is the chance it will rain tomorrow?

The Bayesian approach to probability theory allows the assignment of probabilities to a single event and allows individuals to represent probabilities as beliefs. It is a rich theory, as will be shown later, with approaches to learning and inference congruent with those for developing intelligent systems.

Another reason for not considering probability in AI research is its perceived intractability in complex systems. This perception was credible until the advent, in the 1980s, of graphical models such as Bayesian networks with efficient inference engines [Laskey 1999].

The idea that probability and statistics does not belong in AI research is as wrong as stating there is no place for symbols and logic. Probability does have a niche in AI, as Pearl states

> The aim of artificial intelligence is to provide a computational model of intelligent behavior, most importantly, commonsense reasoning. The aim of probability theory is to provide a coherent account of how belief should change in light of partial or uncertain information. Since commonsense reasoning always applies to incomplete information, one might naturally expect the two disciplines to share language, goals, and techniques [Pearl 1988].

In recent years the AI community has recognized this and is beginning a paradigm shift from more deterministic and ad hoc methods for dealing with uncertainty to approaches based on the first principles of probability. This is evidenced by the establishment of conferences such as Uncertainty in Artificial Intelligence and AI and Statistics. Further, the majority of speakers, including the keynote speaker, at the 1998 combined

International Conference of Machine Learning, Computational Learning Theory, and Uncertainty in Artificial Intelligence were Bayesian [UAI98 1998].

2.1.1 Basic Axioms

A probability measure can be defined in terms of the three basic axioms of probability theory as described by Kolmogorov [Kolmogorov 1956]. Let S denote a set representing all possible outcomes of an uncertain contingency. An *event* is a subset $A \subseteq S$, which represents the proposition that the actual situation is one of the outcomes in the set A. The individual elements of S, denoted by lowercase letters, are called *atoms* or *elementary outcomes*. Singleton subsets {a} are called *atomic* or *elementary events*. These atomic events are assumed to be *mutually exclusive* (no two can occur) and collectively exhaustive (at least one must occur). The axioms of probability are

$$0 \leq P(A) \leq 1 \text{ for every } A$$

$$P(S) = 1$$

$$P\left(\bigcup_{i=1}^{n} A_i\right) = \sum_{i=1}^{n} P(A_i) \text{ if } A_i \cap A_j = \varnothing \text{ and } i \neq j \text{ for all i and j}$$

2.1.2 Bayes' Theorem

An important concept of probability calculus is *conditional probability*. The conditional probability of A given B represents that B has occurred. Conditional probability is defined as

$$P(A|B) = \frac{P(AB)}{P(B)}, \text{ if } P(B) > 0 \tag{2.1}$$

where P(AB) is the probability of both A and B occurring and is known as the joint probability of events A and B. The denominator of (4) is the *marginal probability* of B and is derived for discrete events by summing over all possible states of event A. More formally, $P(B) = \sum_{a \in S} P(B|a)P(a)$. (In a slight abuse of notation we suppress the brackets for the atomic event {a} where the meaning is clear from context).

From (2.1), the definition of the *joint probability* of A and B is

$$P(AB) = P(B)P(A|B) \tag{2.2}$$

where P(AB) is the probability of both events A and B occurred. A generalization of (2.2) for multiple events is a fundamental rule of probability calculus known as the Multiplication Theorem, product rule, or chain rule and is defined as

$$P(A_1 A_2 \ldots A_n) = P(A_1)P(A_2|A_1)\ldots P(A_n|A_1 A_2 \ldots A_{n-1}) \tag{2.3}$$

Finally, applying the product rule to the numerator of (2.1) yields *Bayes' Theorem*.

$$P(A|B) = \frac{P(B|A)P(A)}{P(B)} \tag{2.4}$$

In this expression, P(A|B) is known as the posterior probability of A given B. P(B|A) is called the likelihood, P(A) is the prior probability, and P(B) is the marginal probability of B. The denominator is also known as the normalization constant since it is required in order for all of the events of the posterior to sum to one.

A natural way of thinking of the meaning of Bayes' Theorem is as a ratio of two posterior probabilities. Given events A and $b_1, b_2 \in B$, the posterior odds ratio is defined as $\dfrac{P(A|b_1)}{P(A|b_2)} = \dfrac{P(b_1|A)}{P(b_2|A)} \dfrac{P(A)}{P(A)}$. Notice the normalization constant from (2.4) cancels. The

first ratio on the right is known as the likelihood ratio and the second ratio is the prior odds ratio. The posterior odds ratio is higher if b_1 is more likely than b_2 given A.

Bayes' Theorem is also an important and natural learning rule. P(A) is the probability that A will occur prior to the evidence B. In reality P(A) is conditioned upon the current background knowledge $P(A|\xi)$ where ξ represents background knowledge, the background knowledge is assumed and therefore suppressed for convenience. Let B represent new evidence, where the weight of the evidence is the likelihood P(B|A). Then the new probability of A given the new evidence B is the posterior P(A|B). The posterior can now be used as the prior if additional evidence is received. This form of sequential learning is analogous to how intelligent systems learn and is an important concept in learning theory. This approach to learning will be described in more detail later [De Finetti 1995].

2.1.3 Random Variables

Probability as described above has only propositional expressive power. To use probability to reason about complex tasks, first order expressiveness is needed. In probability and statistics, first order reasoning power is achieved by the concept of a *random variable*. A random variable is a function x where a values x(s) in the set X is defined for every $s \in S$ with probability P [Savage 1972]. I'll use upper-case letters (e.g. X, Y, Z) to represent random variables. When a specific value of a random variable is observed, it is *instantiated*, and I'll represent instantiated variables (specific values) as lower-case letters (e.g. x, y, z). I use bold upper-case letters to represent a set of random

variables (e.g. **X**, **Y**, **Z**) and bold lower-case letters to represent a set of specific values (e.g. **x**, **y**, **z**).

Random variables permit a much more powerful ability to quantify statements than in traditional logic, in which the only quantifications are the universal and existential quantifiers. In probability theory, universal quantification corresponds to probability 1. For example, the rule "For all s, if x(s) then y(s)", represented in logic as $(\forall s)[x(s) \rightarrow y(s)]$, is represented in probability theory as P(y|x)=1 (where the implicit functional relationship on the element s of the sample space is usually suppressed). If it is known only that some x's imply y, logic can say only $(\exists s)[x(s) \rightarrow y(s)]$. Probability theory allows statements about the strength of the relationship, such as P(y|x) = α (about how strongly belief in y is justified given knowledge about x) or $P(x \rightarrow y) = \beta$ (about the probability that material implication holds).

It is interesting to note in this connection that logicians have always been uncomfortable about the mismatch between material implication and human intuition about conditional statements. Statements about conditional probabilities P(y|x) satisfy intuitions about conditionals far better than do statements about material conditionals, P(x→y) [Goodman, Nguyen et al. 1991].

2.1.4 Bayesian Probability and Learning

Bayesian probability theory, also known as subjectivist probability, defines a probability as a rational agent's degree of belief about an uncertain event. If the agent is certain the event, A, will occur then P(A) = 1. Likewise, if the event is impossible then

P(A)=0. Fortunately, most events in life are uncertain, so $0 \le P(A) \le 1$. As the agent's degree of belief increases from impossible to certainty, P(A) increases from 0 to 1. Since the theory applies to any rational agent, the degree of belief associated with any event can vary from agent to agent. If two or more rational agents with different degrees of belief about an uncertain event are provided the same evidence concerning that event, their degrees of belief will converge.

In order for a rational agent's degree of belief to be considered a probability the agent must conform to the *axioms of coherent belief*. These axioms originally put forth by Savage [Savage 1972] are provided by Watson and Buede [Watson and Buede 1987] as follows:

1. For any two uncertain events, A is more likely than B, or B is more likely than A, or they are equally likely
2. If A_1 and A_2 are two mutually exclusive events, and B_1 and B_2 are any other mutually exclusive events; and if A_1 is not more likely than B_1, and A_2 is not more likely than B_2; then (A_1 and A_2) is not more likely than (B_1 and B_2). Further, if either A_1 is less likely than B_1 or A_2 is less likely than B_2, then (A_1 and A_2) is less likely than (B_1 and B_2).
3. A possible event cannot be less likely than an impossible event.
4. Suppose A_1, A_2,... is an infinite decreasing sequence of events; that is, if A_i occurs, then A_i occurs for any i. Suppose further that each A_i is not less likely than some other event B, again for any i. Then the occurrence of all the infinite set of events A_i, i=1...∞, is not less likely than B.
5. There is an experiment, with a numerical outcome, such that each possible value of that outcome, in a given range, is equally likely.

DeGroot, [DeGroot 1970], proved that the axioms of coherent belief implies that for every event A there is a unique probability P(A) that satisfies the basic axioms of probability theory.

The Bayesian view of probability is a natural formalism for AI. With the Bayesian approach it is quite easy for a system to provide the probability for a single event. It is simply the agent's prior probability for that event. Or if provided some evidence, it is the agent's posterior probability given the evidence.

This discussion leads to the concept of learning. The Bayesian construct for learning is quite similar if not synonymous with that of an intelligent agent. With the Bayesian approach to learning, a rational agent begins with a prior belief of an event A. The prior, as mentioned earlier, is developed from background knowledge of the event. Learning occurs when experience is accumulated in the form of information. The information is associated with the likelihood and taken together with the prior in Bayes' Theorem and the posterior is updated to reflect the new information.

Let's look at an example of flipping an unfair coin that is biased in an unknown manner. The problem is to find the parameter that allows us to predict whether the next flip will land heads or tails. Using Bayes' Theorem

$$P(\theta|F) = \frac{P(F|\theta)P(\theta)}{P(F)}$$

where θ is the parameter and F is the result of the flip. Since this is a binomial problem, the likelihood function $P(F|\theta)$ can be represented as a binomial distribution.

$$P(F|\theta) = \binom{n}{h}\theta^h(1-\theta)^{n-h} \tag{2.5}$$

where h is the number of heads and n is the total number of flips. A convenient distribution for the prior is a *beta* distribution:

$$P(\theta) = Beta(\theta|\alpha, \beta) = \frac{\Gamma(\alpha + \beta)}{\Gamma(\alpha)\Gamma(\beta)} \theta^{\alpha-1}(1-\theta)^{\beta-1} \qquad (2.6)$$

where $\alpha > 0$ and $\beta > 0$, and $\Gamma(\cdot)$ is the *Gamma* function defined as $\Gamma(x+1) = x\Gamma(x)$ and $\Gamma(1) = 1$. The parameters α and β are known as the hyperparameters of the prior.

Notice the similarity of the Beta distribution and the binomial distribution. Let the hyperparameter α represent the prior number of heads observed and β the prior number of tails observed. Taking the product of the likelihood and prior yields the beta posterior distribution

$$P(\theta|F) = \frac{\Gamma(\alpha + \beta + n)}{\Gamma(\alpha + h)\Gamma(\beta + n - h)} \theta^{\alpha + h - 1}(1-\theta)^{\beta + n - h - 1} \qquad (2.7)$$

The beta prior distribution is a *natural conjugate distribution* of the binomial distribution. A conjugate distribution is defined as a class of distributions P when used as a prior with a likelihood of the class of distribution F yields a posterior of the class P. This is not only convenient for computational outcomes but also the prior hyperparameters are valuable sources of information as in the prior counts of heads and tails.

To continue the example, if the prior distribution of the number of heads or tails is unknown, a common approach is to assume $\alpha = 1$ and $\beta = 1$. This produces a Uniform distribution between $[0,1]$ and is known as a vague prior since the agent's prior knowledge is ignorance about the parameter θ. If the agent has knowledge of the weight of the coin, then an appropriate assignment can be made to the hyperparameters to reflect this knowledge.

The prediction that the next toss will result in a head is the expectation of θ with respect to the beta distribution of equation (2.7). The expectation of this beta distribution is simply

$$P(F_{n+1} = heads|F_1...F_n) = \frac{\alpha + h}{\alpha + \beta + n}$$

Suppose the agent uses the vague prior and after 15 flips of the coin 10 have landed heads. The probability the next flip will result in a head is $P(F_{n+1}|F_1...F_n) = \frac{11}{17} \approx 0.65$

[Bernardo and Smith 1994] [Heckerman 1996].

2.1.5 Conditional Independence

When A and B in equation (2.4) are *independent events*, P(A|B) = P(A) since knowledge of B has no bearing on the likelihood of event A. Thus from (2.1), P(AB)=P(A)P(B). When two events are not independent they are said to be *dependent events* [Bain and Engelhardt 1992].

It is possible and in fact a very common occurrence for two events, A and B, to be dependent, but when a third event is observed, the two events A and B become independent. Suppose there is empirical evidence that children from wealthy families as a group perform well in high school. In addition, there is even stronger, independent evidence that children who perform well in high school perform well in college. Let Harry be from a wealthy family. From the evidence one would believe rather weakly that Harry will perform well in college. However, if one knew that Harry performed well in high school, the knowledge he is from a wealthy family is irrelevant to one's belief that

he will perform well in college. This common phenomenon is known as conditional independence.

Numerically, conditional independence is represented as the factorization of the conditional joint distribution of (X,Y) given Z

$$P(X,Y|Z) = P(X|Z)P(Y|Z) \qquad (2.8)$$

Another equivalent measure is

$$P(X|YZ) = P(X|Z) \qquad (2.9)$$

The conditional joint distribution, (2.8), can be interpreted as saying that given Z, the joint probabilities of X and Y are independent conditional probabilities of each event. While (2.9) states that once Z is known, knowledge of event Y is irrelevant. Conditional independence allows us to simplify the intractable full joint probability density when no conditional independence is considered. For instance, suppose we have four binary variables A, B, C, and the observed variable D. The full joint probability density is $P(ABC|D) = P(A|D)P(B|AD)P(C|ABD)$. If it is known that B is independent of A given D and C is independent of A given D, then $P(ABC|D) = P(A|D)P(B|D)P(C|BD)$ reducing the number of probabilities required from 30 to 14.

The qualitative descriptions in (2.8) and (2.9) entail a number of important qualitative relationships that accord with intuitions about independence. Recently, Dawid, Pearl, and others have developed qualitative axioms formalizing properties of conditional independence in more qualitative terms. This new calculus of conditional independence leads to a more intuitive understanding and allows development of useful applications within and outside probability theory.

For this work, I'll follow the notation of Pearl and indicate the fact that X is independent of Y given Z by I(X,Y,Z) [Pearl 1988]. Dawid proposed the following five properties of conditional independence. I'll use the form $W \leq Y$ to indicate that W is a function of Y

P1: I(X,Z,Y)	$\Rightarrow$	I(Y,Z,X)
P2: I(X,X,Y)		
P3: I(X,Z,Y) $W \leq Y$	$\Rightarrow$	I(X,Z,W)
P4: I(X,Z,Y) $W \leq Y$	$\Rightarrow$	I(X,(WZ),Y)
P5: I(X,Z,Y) & I(X,(YZ),W)	$\Rightarrow$	I(X,Z,(Y,W))

P1 states that conditional independence is symmetric. If X is independent of Y given Z then Y is independent of X given Z. P2 is straightforward, if X is known, then Y is irrelevant for knowledge of X. P3 states that if X is independent of Y given Z and W is a function of Y then X is also independent of W given Z. P4 is a little subtler, if X is independent of Y given Z and W is a function of Y, then X is still independent of Y given both Z and W. Finally, P5 states that if X is independent of Y given Z and X is independent of W given Y and Z then X must be independent of Y and W given Z [Dawid 1979], [Dawid 1980], [Pearl 1988]. The intuition behind these properties will become clearer in the next section, which discusses a formalism called graphical models for representing conditional independence relationships.

2.2 Graphical Models

Graphical models are a general class of mathematical models that allow us to visually represent relationships among variables. Graphical models can be used to construct joint probability distributions over large sets of random variables by using modular components that involve only a few variables at a time. There is a mathematical and statistical theory that underlies graphical models [Lauritzen 1996]. This theory provides computationally tractable methods for drawing inferences about the relationships among the variables encoded in the model. A graph consists of a set of vertices (also called nodes) and edges: G-(V,E). G is a *complete graph* if all its vertices are connected by an edge. A graph H is a subgraph of G if the vertices of H are a subset of the vertices of G, $V_H \subseteq V_G$, and the edges in H are a subset of the edges in G, $E_H \subseteq E_G$. A *clique* is a maximal complete subgraph. A connected graph with no closed paths is known as a *tree* [van Lint and Wilson 1996].

In graphical models the vertices represent random variables and the edges represent relationships among the variables. The edges can be directed or undirected. A directed edge connects an ordered pair of vertices (I, J) and is graphically represented by an arrow pointing from the vertex I to the vertex J. The set of all parents of J is represented as pa(J) while the set of all children of I is represented by ch(I). For example, in Figure 2.1, the set pa(D) = {B,C} while the set ch(B) = {C,D}.

A vertex I with a link to vertex J is said to be a neighbor of J or ne(J). The set of all parents and neighbors of a vertex is its boundary, bd(J). The neighbors of B in Figure 2.2

are ne(B) = {A,C,D}. For undirected graphs the neighbors of a vertex are the same as its boundary since there are no parents of vertices.
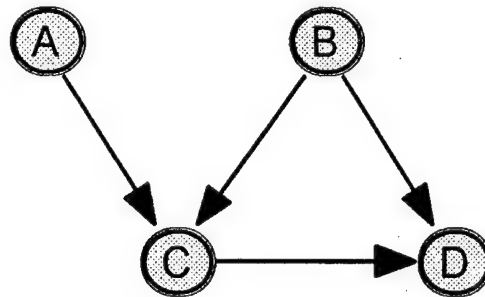
Figure 2-1 A Simple Directed Graph
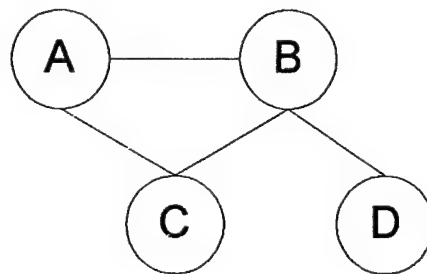
Figure 2-2 A Simple Undirected Graph

When there exists no edge between two nodes the corresponding variables are independent given the remaining variables. The resulting undirected graph is called a conditional independence graph or Markov network. Markov networks must obey the following three Markov properties as described by Lauritzen [Lauritzen 1996].

pairwise Markov property: pairs of non-adjacent nodes are independent given the remaining variables.

local Markov property: given evidence of all adjacent nodes, any node is independent of all the remaining nodes.

global Markov property: any two subsets of nodes separated by a third subset of nodes is conditionally independent given the values of the third subset of nodes.

Using the pairwise Markov property, the network in Figure 2.2 is specified by the conditional independence statements: I(D,(B,C),A) and I(D,(A,B),C). In words, variable D is independent of variable A given variables B and C. The second term states that variable D is independent of variable C given variables A and B. It's easy to see that these statements also apply for local Markov property and global Markov property. From the network in Figure 2.2, it is trivial to see that if node B and its edges were removed, node D would be separated from the set {A,C}. Intuitively, the observation is implied by the global Markov property that I((A,C),B,D). The Separation Theorem formalizes this implication.

> **Theorem 2.1 The Separation Theorem [Whittaker 1990]**. If **A**, **B**, and **C** are vectors containing disjoint subsets of variables from X and if, in the Markov network of X, each node in **B** is separated from each node in **C** by the subset **A**, then I(**B,A,C**).

Applying the Separation Theorem to the network of Figure 2.2, the conditional independence statements given above are reduced to I(D,B,A) and I(D,B,C) [Lauritzen 1996], [Whittaker 1990], and [Pearl 1988].

From the graphical representation of Markov networks it is easy to determine whether sets of variables in the network are conditionally independent given another set

of variables. It is also easy to construct the Markov network from a set of conditional independence statements.

Once conditional independence relationships have been encoded in the graph structure, the next step in building the network is to assign probabilities to the network. Let G represent the Markov network. Identify the cliques of G. Assign a nonnegative potential function for each clique C_i, namely $g_i(c_i)$. The potential function of a clique is a function of the joint probability of the variables in the clique. Potentials represent the strength of belief in a configuration. Increasing the potential of one clique while holding the remaining potentials constant raises the probability of that configuration and decreases the probability of the others. The joint probability represented by the network is $P(X_1 \ldots X_n) = K \prod_{X_1 \ldots X_n} g_i(c_i)$ where K is the normalization constant.

If the network is decomposable it can be converted to a join tree. A join tree is a tree where the vertices are the cliques of the graph and the edges connect cliques with common variables from the original network. For example the cliques in Figure 2.2 are {ABC} and {BD}. The vertices of the join tree are the cliques. There is one common variable, namely B, among the two cliques. There is an edge connecting the two cliques. So the join tree is [ABC]-[BD]. Applying the independence assumptions to the joint distribution derives a set of potential functions for the network.

$$P(ABCD) = P(A)P(B|A)P(C|AB)PD(ABC)$$

$$P(ABCD) = P(A)P(B|A)P(C|AB)PD(D|B)$$

$$P(ABCD) = P(ABC)\frac{P(BD)}{P(B)}$$

The potential functions are $g(ABC) = P(ABC)$ and $g(BD) = \dfrac{P(BD)}{P(B)}$. These potential

functions are not unique. For example, the potential functions $g(ABC) = \dfrac{P(ABC)}{P(B)}$ and

$g(BD) = P(BD)$ or any constant multiple of these are just as valid.

The concept of clique potentials in general can be problematic. This is especially a problem when assignment is based on expert judgement. However, when a graph is directed and acyclic (i.e. a Bayesian network), the distribution is specified by the conditional probability of each variable given its parents. This is a meaningful assignment for domain experts. A typical task for a Bayesian network is to compute the probability of some variables given other variables. In general, this is a NP-hard problem [Cooper and Herskovits 1992]. When the network is singly connected, meaning there are no cycles (directed or undirected), inference is linear in the number of nodes. Algorithms have been developed to transform multiply connected Bayesian networks into trees of cliques (i.e. junction trees) and propagate (i.e. do inference) on the transformed graph [Spiegelhalter, Dawid et al. 1993], [Pearl 1988].

2.3   Bayesian Networks

2.3.1   Introduction

Section 2.2 introduced graphical models. In this section we focus on an important subclass in which the graph is directed with no directed cycles, called Bayesian networks. Bayesian networks are important because they provide a modular way to represent units

of probabilistic knowledge ("probabilistic rules"). The graphical metaphor makes the model understandable.

Probably the easiest way to describe a Bayesian network is through an example. Imagine the reasoning process you may have if you visit a friend's home and begin to sneeze. You may conjecture that you are coming down with a cold or perhaps your allergies are beginning to act up. But you are only allergic to cats, so you look around for signs that your friend has a cat. You may look for the cat or maybe inspect the furniture for scratch marks. Figure 2.3 shows a Bayesian network that models your chain of reasoning.
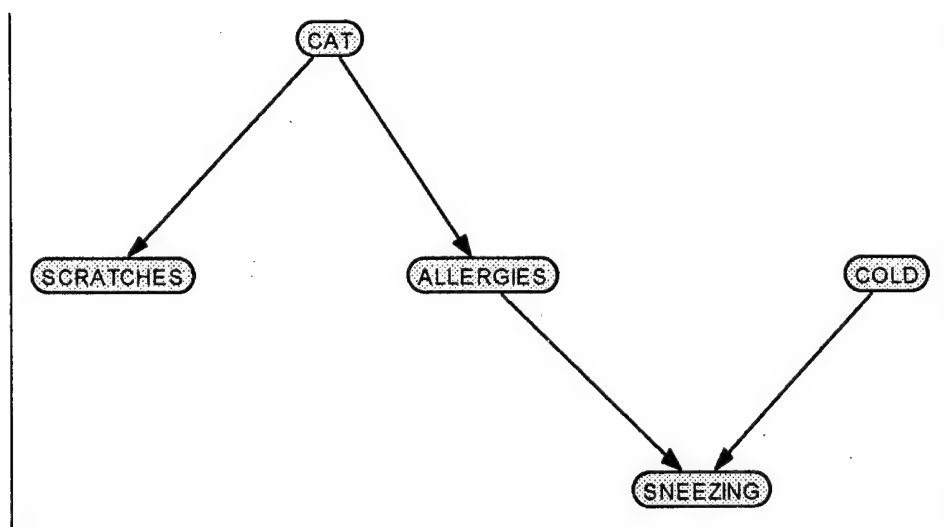
Figure 2-3 Sneezing Example

From Figure 2.3, one can see that the qualitative relationships among variables in a Bayesian network are represented as a directed acyclic graph. The nodes of the graph represent random variables. The arcs represent dependencies between variables. The graphical representation of a Bayesian network is known as the *structure* of the network.

One way to think of it is that an allergy or a cold may cause sneezing. More importantly, the lack of an arc represents conditional independence. In other words, if the values of the parents of a variable are known, the other variables in the network are irrelevant in determining the values of the variables under consideration. For example, if you know your friend owns a cat, no other variable in the model effects your belief of whether there are scratches on the furniture. Yet if you do not know your friend owns a cat, observing whether you are sneezing or knowing whether you have a cold will change you beliefs about scratches on the furniture.

Formally, in this model scratches and allergies are conditionally independent give the presence of a cat, which we denote by I(SC,CT,A) where SC represents variable scratches, A represents the variable allergies and CT represent the variable cat. In directed graphical models, conditional independence is also called d-separation. If I(A,C,B) we say C d-separates A from B. The variable cat d-separates the variables scratches and allergies. There are three possible configurations for d-separation. If three variables are connected in serial, A->B->C, without any additional arcs between predecessors and ancestors (e.g. X->A->B->C->Y where there is also an arc from X to Y) then I(A,B,C). A set of variables are also d-separated if the connections are diverging and the value of B is known as in A<-B->C, then I(A,B,C). The next configuration is a little subtler. If the connections are converging, A->B<-C, then I(A,B,C) if and only if B or its descendants have not been observed. For instance, in Figure 2.3 if you are not sneezing, then observing the scratches on the furniture only influences your belief that your allergies may act up and has no effect on your belief whether you have a cold.

However, if you are sneezing then you may have allergies, a cold, or both. Observing scratches on the furniture will increase your belief that you have allergies and decrease your belief you have a cold. So allergies are d-separated from cold when sneezing is not observed. The concept of d-separation can be used to deduce conditional independence among variables in a Bayesian network without consideration of the numerical probabilities.
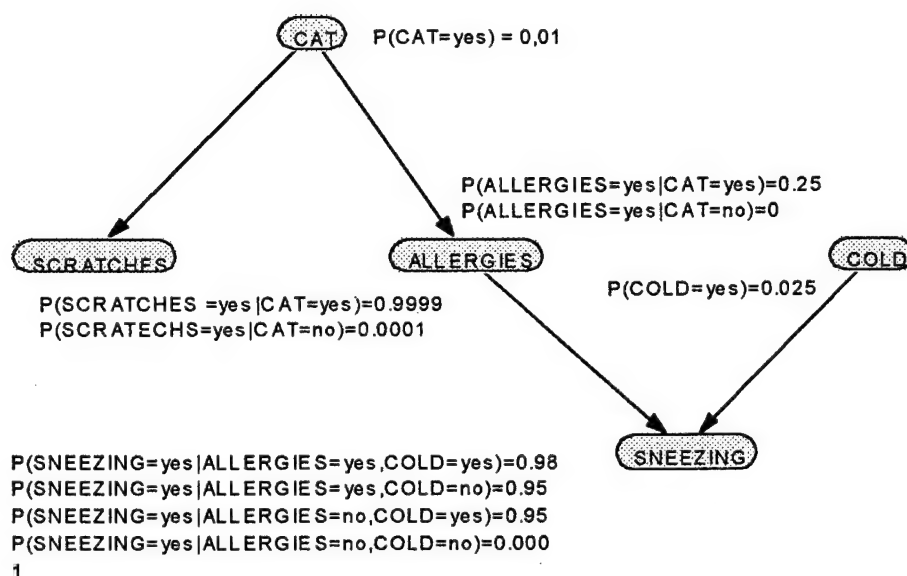
Figure 2-4 Sneezing Example (Structure and Parameters)

Beliefs are represented as probabilities in Bayesian networks. Each node has associated with it a conditional probability table (CPT) for discrete variables or conditional marginal functions for continuous variables. The values of the CPT are prior probabilities for root nodes and conditional probabilities of the remaining nodes conditional upon their parents. The probabilities of a Bayesian network are known as the

parameters of the network. Figure 2.4 shows the structure and parameters for the sneezing example. Network parameters can also be viewed in terms of their marginal values. Most Bayesian network software packages allow the structure and marginal values to be viewed on one graph. Figure 2.5 shows the network and marginal values for the sneezing example.
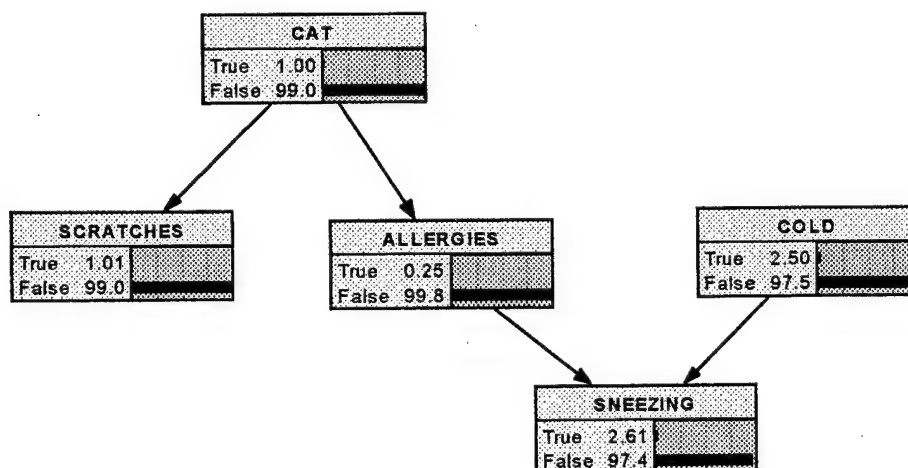
Figure 2-5 Sneezing Example (Structure and Marginal Probabilities)

One of the many advantages of Bayesian networks is that they efficiently encode the joint probability of the set of variables, $\mathbf{X} = \{X_1, X_2, ..., X_n\}$. If a fully general probability model of a binary valued set of n variables were specified, 2^n probabilities would be needed to specify the joint distribution of n binary variables. If the binary-valued model has five variables, as in the sneezing example, then 2^5 or 32 probabilities are necessary. If the model has 10 variables, 1024 probabilities are needed. For a moderate size model of 50 binary-valued variables 2^{50} or approximately 10^{14} probabilities are required and if each of the 50 variables have three possible values there will be over

10^{23} probabilities. A pretty daunting data entry and computation task to say the least. Recall, the joint probability can be represented as

$$P(X_1 X_2 ... X_n) = P(X_1)P(X_2|X_1)...P(X_n|X_1 X_2 ... X_n) \qquad (2.10)$$

By taking into account the conditional independence properties each of the terms in (2.10) need involve only the parents of the node. Therefore, the number of probabilities needed is reduced significantly. For the sneezing example in Figure 2.4, the joint probability is

$$P(CT, SC, A, CO, SN) = P(CT)P(SC|CT)P(A|CT)P(CO)P(SN|A, CO)$$

where CT=cat, SC=scratches, A=allergies, CO=cold, and SN=sneezing. The number of probabilities needed is only 20 compared to 32 for the full probability model. For larger models involving variables with many values the savings in numbers of parameters needed to specify a Bayesian network versus a full probability model can be on the order of hundreds or millions depending on the structure of the network. It is easy to see from the sneezing example that the general formula for the joint distribution is expressed as

$$P(\mathbf{U}) = \prod_i P(X_i|pa(X_i)) \qquad (2.11)$$

where $\mathbf{U} = \{X_1, X_2, ..., X_n\}$ and $P(X_i|pa(X_i))=P(X_i)$ when $pa(X_i)=\varnothing$.

The power of Bayesian networks is not just parsimony, but also their ability to reproduce qualitative patterns of plausible reasoning for inference tasks of much greater complexity than formerly thought to be feasible for computer programs. Reasoning in Bayesian networks is made possible through the inference algorithms discussed briefly in Section 2.2. Although these algorithms are essential for inference in Bayesian networks,

the scope of this research is not concerned with how these algorithms work. Therefore, I will not discuss how probabilities are updated, but I will provide a demonstration using the sneezing example. Readers interested in understanding the details of Bayesian network inference algorithms should see [Pearl 1988], [Jensen 1996], [Spiegelhalter, Dawid et al. 1993], [Neapolitan 1989], and [Jordan 1999].
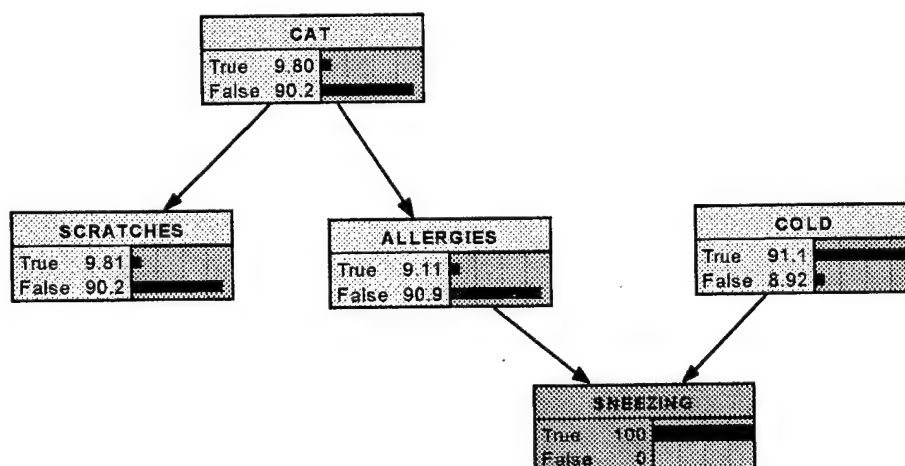
Figure 2-6 Sneezing Example with evidence of sneezing

Referring to the sneezing example in Figure 2.3, you go to your friend's home for a quick visit. After a few minutes in the house you begin to sneeze. At first you believe that you must be catching a cold. Figure 2.6 shows the network with the observation that you are sneezing. I will use the terms observation, evidence, and instantiation interchangeably. From Figure 2.6, the probability (your belief) that you are catching a cold has increased significantly from the prior probability. Also note that the probability your allergies are acting up have also increased, but not as much. Clearly, you believe you are catching a cold. Now suppose that as you sit on your friend's couch you notice

scratches on the furniture. The updated probabilities for this observation are shown in Figure 2.7.
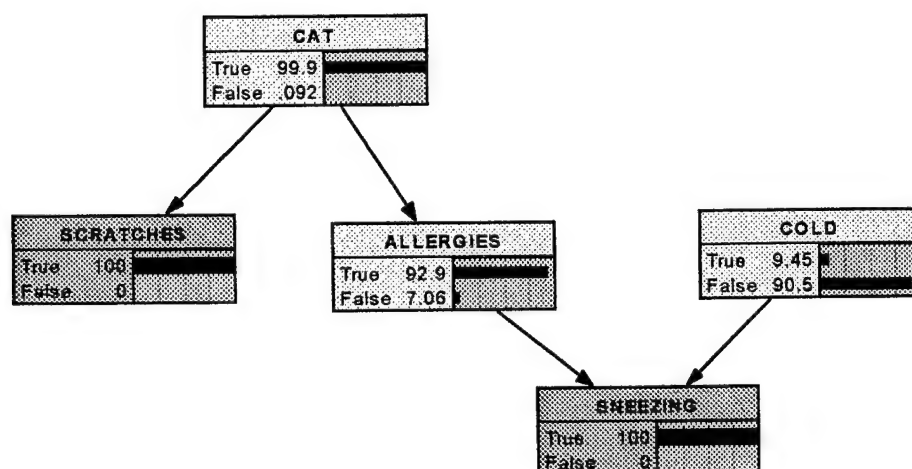
Figure 2-7 Sneezing Example with evidence of sneezing and scratches

You realize that you are not getting a cold, but that your friend has a cat and your allergies are causing you to sneeze. Notice that the probability that you have a cold decreased with this new evidence and the probability that you have an allergy increased significantly. This phenomenon is called *explaining away*. The new evidence of a cat causes you to believe you have allergies and has explained away the sneezing, so the cold hypothesis no longer has as much support. You probably have allergies and not a cold (though there is still a small chance you have a cold). Explaining away is a powerful reasoning property that is inherent in Bayesian networks. Other AI applications, such as rule-based systems, require all the exceptions be listed then somehow ranked in an ad hoc way and tweaked until reasonable answers are obtained. This all arises naturally as a

consequence of conditional independence assumptions and relative magnitudes of the probabilities in a Bayesian network model. Thus, the phenomenon of explaining away can be explained from first principles and no ad hoc devices are needed.

In the example, the evidence of sneezing propagated throughout the network causing the remaining variables to update their probabilities. This is not always the case. Take for example, the fact that you know your friend has a cat but have not started sneezing. The network is shown in Figure 2.8. Observe from this example that the probability that you will soon start sneezing has increased along with the probability that your allergies will act up and there are scratches on the furniture. Yet the probability that you have a cold is unchanged. This is because allergies and cold are independent (there is no arc connecting them) unless a dependency is introduced by obtaining evidence about sneezing.
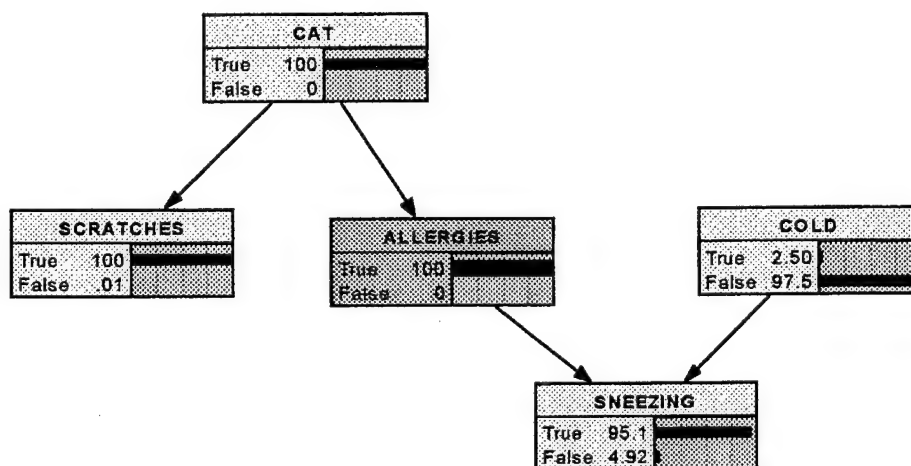
Figure 2-8 Sneezing Example with evidence of allergies

2.3.2 Why Bayesian Networks

This brief section will touch on some of the advantages of using Bayesian networks, especially in artificial intelligence research and applications. The first and arguably most important advantage is their ability to reason under uncertainty. The sneezing example demonstrated how one could draw inferences from a model as we gather information. In this example, when you began to sneeze, you were uncertain as to whether you had a cold or allergies but you believed you had a cold. As you gathered more information, your beliefs about your having a cold decreased and that your allergies were acting up increased.

One of the principal characteristics of an intelligent system is the ability to classify and predict. Unlike many AI approaches, Bayesian networks allow classification and prediction from the same model. This is because reasoning, in Bayesian networks, propagates in all directions. Whereas, in decision trees, rule-based systems, and many neural networks, inference is in one direction so the model is either able to classify or predict but not both.

Further, Bayesian networks process information either batch or serial. Each piece of information about some variable causes the network to update the probabilities of the other variables in the network. This is important since complete information is not always at hand to make decisions. One need not specify in advance which variables will become known during the course of reasoning and which will be the target of queries. The same network can be used to infer sneezing from scratches on the furniture or to infer scratches on the furniture from sneezing.

Finally, and most definitely not the last advantage, Bayesian networks provide probability distributions (or beliefs) for all variables in the network. This is especially important in expert systems. With the distributions available one can see how likely each value is to occur for each variable. In logic systems and most common neural networks and classification trees, only a single response is provided for a query. When more than one response is just as probable, within some limit, a single response is not satisfactory. It may be important to know the amount of uncertainty associated with responses. The sneezing example clearly shows that conclusions provided by Bayesian networks encode the uncertainty in the decision. For instance, when you observed sneezing there were beliefs associated with cold and allergies.

2.4 The Problem of Learning Bayesian Networks from Data

The most common approach to building Bayesian networks is to elicit knowledge from an expert. This is fine for smaller networks, but when the number of variables becomes large, elicitation can become a tedious and expensive affair. There may also be situations where the expert is either unwilling or unavailable. Whether or not experts are available, if there are data it makes sense to use it in building a model for a problem.

The learning problem can be broken into two components: learning the structure, B_S, and learning the parameters, B_P. If the structure is known[1] then the problem reduces to learning the parameters. If the structure is unknown, the learner must first induce the structure before learning the parameters. In addition, it may be important to learn if the

---

[1] The true structure may not really be known in a complex problem. I use the term known to be consistent with current literature and use of the term. A more accurate term is the structure is "given" or "provided."

network has any hidden variables and, if so, how many and where. Finally, the data used in learning may contain missing values. Figure 2.9 shows the learning problem broken into four components categorized by difficulty.

	Known Structure	Unknown Structure
Complete Data	Easy	Manageable
Incomplete Data	Manageable	Hard

Figure 2-9 Learning Categories

2.4.1 Complete Data and Known Structure

The easiest learning category is to learn the parameters from a known structure and complete data. Complete data in this sense refers to data where all the cases contain values for all the variables. In addition, I assume there are no hidden variables and the cases are independent samples from the same probability distribution. The goal of learning is to estimate the parameters of this common distribution.

Recall from section 2.1, the Bayesian approach to learning combines prior knowledge of the distribution with the observed data to learn an updated probability or belief. For the discrete case, the variables are multinomial and the values of each variable are conditional upon the sets of values of the parents of each variable. Let θ_S be the vector of variable parameters, D represents the data, and B_S is the given structure. Let θ_i denote the vector of parameters for variable X_i. The values for X_i are conditional on

the configurations of values of the parents of X_i (recall from Section 2.3, the CPT for a variable, X_i, in a Bayesian network is $P(Xi|pa(Xi))$). Further, let the vector of parent configurations for X_i be defined as $\mathbf{pa}_i = \{\mathbf{pa}_i^1, \ldots, \mathbf{pa}_i^{q_i}\}$ where q_i is the total number of possible parent configurations. Let $\theta_i = \left(\left(\theta_{ijk} \right)_{k=1}^{r_i} \right)_{j=1}^{q_i}$, where r_i are the total number of values from X_i, be the parameter vector for X_i. For convenience, the parameter vector, θ_i, can be represented as a vector for each parent configuration, $\theta_i = \{\theta_{ij1}, \ldots, \theta_{ijr_i}\}$. Assuming the data are complete then

$$P(\theta_s | D, B_s) = \prod_{i=1}^{n} \prod_{j=1}^{q_i} p\left(\theta_{ij} | D, B_s \right) \tag{2.12}$$

Further, assuming the parameters are independent they can be estimated separately. From Bayes' Theorem

$$P\left(\theta_{ij} | D, B_s \right) = \frac{P\left(D | \theta_{ij}, B_s \right) P\left(\theta_{ij} | B_s \right)}{P(D)} \tag{2.13}$$

the likelihood function is the multinomial distribution

$$P\left(D | \theta_{ij}, B_s \right) = \binom{N_{ij}}{N_{ij1}, N_{ij2}, \ldots N_{ijr_i}} p_1^{N_{ij1}} p_2^{N_{ij2}} \cdots p_{r_i}^{N_{ijr_i}} \tag{2.14}$$

where the N_{ijk} are the counts for variable X_i, parent configuration $pa_j(X_i)$, and value $X_i = k$. The term $N_{ij} = \sum_{k=1}^{r_i} N_{ijk}$. The natural conjugate of a multinomial distribution is a Dirichlet distribution (a Dirichlet distribution is the multinomial generalization of the beta distribution).

$$P(\theta_{ij}|B_s) = \frac{\Gamma(\alpha_{ij})}{\Gamma(\alpha_{ij1})\cdots\Gamma(\alpha_{ijr_i})} p_1^{\alpha_{ij1}-1}\cdots p_{r_i}^{\alpha_{ijr_i}-1} \qquad (2.15)$$

where α_{ijk} are the prior counts and $\alpha_{ij} = \sum_{k=1}^{r_i}\alpha_{ijk}$. The resulting posterior distribution is a

Dirichlet distribution of the form

$$P(\theta_{ij}|B_s) = \frac{\Gamma(\alpha_{ij}+N_{ij})}{\Gamma(\alpha_{ij1}+N_{ij1})\cdots\Gamma(\alpha_{ijr_i}+N_{ijr_i})} p_1^{\alpha_{ij1}+N_{ij1}-1}\cdots p_{r_i}^{\alpha_{ijr_i}+N_{ijr_i}-1} \quad (2.16)$$

Luckily, we only need compute the expectation for each value, which is

$$E\left(\theta_{ijk}|P(\theta_{ij}|D,B_s)\right) = \frac{\alpha_{ijk}+N_{ijk}}{\alpha_{ij}+N_{ij}} \qquad (2.17)$$

assuming a vague prior, we can set the prior counts to 1 and (2.17) further reduces to

$$E\left(\theta_{ijk}|P(\theta_{ij}|D,B_s)\right) = \frac{N_{ijk}+1}{N_{ij}+r_i} \qquad (2.18)$$

So the parameter estimation problem simply involves counting the values for each variable and its associated parent configuration, pa(X$_i$) [Heckerman 1996].

2.4.2 Complete Data and Unknown Structure

A more difficult problem is learning the structure of a network given complete data. This problem has received much attention since Cooper and Herskovits introduced an algorithm in 1992 [Cooper and Herskovits 1992]. Learning structure is much different from learning the parameters. The problem consists of searching over potential structures for the "best" structure given the data. This search space can be quite large. Robinson shows that the number of possible structures increases dramatically as the number of

nodes in the network increases. He derives the following recursive function for determining the number of possible structures

$$f(n) = \sum_{i=1}^{n} (-1)^{i+1} 2^{i(n-i)} f(n-i) \qquad (2.19)$$

where n is the number of nodes [Robinson 1977]. So the number of possible structures when there are 2 nodes is 3; for 5 nodes it is 29,000; and for 10 nodes the possible number of structures is approximately 4.2×10^{18}. It is easy to see that the search space of possible structures gets pretty huge as the number of nodes increases. Obviously an exhaustive search is not possible for larger networks, so we need a heuristic algorithm.

There are two components to Bayesian network learning algorithms: scoring metrics and search. The scoring metric measures how well the structure fits the data. The search looks for the structures with the "best" scores.

The scoring metric for network structures can be derived from Bayes' Theorem

$$P(B_s|D) = \frac{P(D|B_s)P(B_s)}{P(D)} \qquad (2.20)$$

We can ignore P(D) since it does not depend on the structure. We can further reduce the problem by making use of the noninformative prior $P(B_s) \propto 1$. If we have prior evidence that certain structures are more likely than others we can use informative prior probabilities for $P(B_s)$.

The problem is now reduced to finding the structure with the maximum likelihood, $P(D|B_s)$. In other words, given a structure, how probable is it that the data were generated from that structure. Fortunately, Cooper and Herskovits and Heckerman et al., have obtained a closed form solution for $P(D|B_s)$ using the formula

$$P(D|B_S) = \prod_{i=1}^{n} \prod_{j=1}^{q_i} \frac{\Gamma(\alpha_{ij})}{\Gamma(\alpha_{ij} + N_{ij})} \prod_{k=1}^{r_i} \frac{\Gamma(\alpha_{ijk} + N_{ijk})}{\Gamma(\alpha_{ijk})} \tag{2.21}$$

where n is the number of variables, r_i is the possible states for variables X_i, q_i is the possible parent configurations $pa(X_i)$, N_{ijk} is the number of observations where X_i=k and $pa(X_i)$=j, $N_{ij} = \sum_{k=1}^{r_i} N_{ijk}$, α_{ijk} are the prior count observations where X_i=k and $pa(X_i)$=j, and $\alpha_{ij} = \sum_{k=1}^{r_i} \alpha_{ijk}$. When the prior counts are set to one (a vague prior) then (2.21) reduces to

$$P(D|B_S) = \prod_{i=1}^{n} \prod_{j=1}^{q_i} \frac{(r_i - 1)!}{(N_{ij} + r_i - 1)!} \prod_{k=1}^{r_i} N_{ijk}! \tag{2.22}$$

Equation (2.21) and (2.22) are known as the Bayesian Dirichlet, BDe, metrics [Cooper and Herskovits 1992], [Heckerman, Geiger et al. 1995], [Fung and Crawford 1990].

Let's look at a simple example taken from Cooper and Herskovits [Cooper and Herskovits 1992]. Table 2.1 is a database with three binary-valued variables. Suppose that we are given the structure in Figure 2.10. Applying equation (2.22) for the structure in Figure 2.10 and the data in Table 2.1, we get.

$$P(D|B_S) = \frac{(2-1)!5!5!}{(10+2-1)!} \frac{(2-1)!1!4!}{(5+2-1)!} \frac{(2-1)!4!1!}{(5+2-1)!} \frac{(2-1)!0!5!}{(5+2-1)!} \frac{(2-1)!4!1!}{(5+2-1)!}$$

$$P(D|B_S) = 2.23 \times 10^{-9}$$

The learner can now use a greedy search technique where we begin by calculating the score for a network with no arcs. In the example of Table 2.1, the score of the network with no arcs is 5.63×10^{-11}. The learner then finds new structures by systematically

adding, deleting, or reversing arcs and calculating the score of the new network. Adding the arc from x1 to x2 raises the score to 1.45×10^{-10}, or a factor of about 2.5 to 1. Adding the second arc, from x2 to x3, gives a score of 2.23×10^{-9} a 15 to 1 factor increase. With each iteration, if the generated network scores higher it is kept as the new network, else the algorithm keeps the previous best network.

The greedy search algorithm proposed by Cooper and Herskovits is K2 [Cooper and Herskovits 1992]. This algorithm takes advantage of the fact that a directed acyclic graph specifies a node ordering. K2 requires the user to provide a node ordering to guarantee a legal structure. It begins by finding the local score for the first node. Then it scores the second node with no arcs. Next it adds an arc from node 1 to node 2 and finds the score for node 2 with the arc. If the score with the arc is greater than the score for node 2 without an arc it keeps the arc, else it removes the arc and proceeds to node 3. Next node 3 is scored with no arcs. Then the algorithm finds the arc from nodes one and two with the best score. If that score is better than the node without an arc it keeps the arc, and tries to add another arc, else it stops searching. This process continues until the final node has been searched. Since there are 2^{n-1} possible parent combinations, the user is required to specify a limit on the number of parents a node is allowed to have to reduce computation time.

One of the drawbacks of K2 is that the user must provide a node ordering. Given n nodes there are n! possible node orderings. Larranaga et al., used a genetic algorithm in combination with K2 to find network structures from complete data [Larranaga, Poza et al. 1996]. The genetic algorithm presented node orderings to K2 and K2 in turn returned

a score as the fitness function for the genetic algorithm. In their study, Larranga et al., compared four types of genetic algorithms. Their algorithm was able to find very high

Table 2-1 A Simple Database

Case	x1	x2	x3
1	present	absent	absent
2	present	present	present
3	absent	absent	present
4	present	present	present
5	absent	absent	absent
6	absent	present	present
7	present	present	present
8	absent	absent	absent
9	present	present	present
10	absent	absent	absent
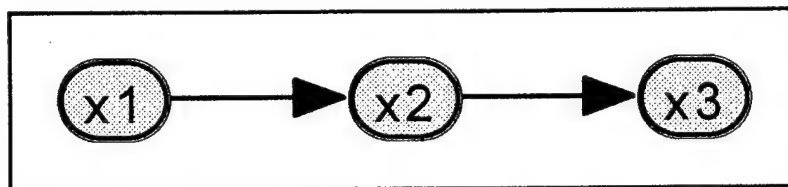
Figure 2-10 Example Structure for Network

Bayesian Dirichlet scores. In some cases, they found networks with higher Bayesian Dirichlet scores than the "true" network that generated the data. Intuitively, these results should raise one's brow. Unfortunately, Larranga et al., only stated the results as fact and did not elaborate further. I will address this issue in Chapter 6.

2.5   Dealing with Incomplete Data and Hidden Variables

Incomplete data is an important source of uncertainty in reasoning problems. If complete data are available, then deterministic algorithms are sufficient. However, most real-world decisions are made with incomplete data at hand. For this research, incomplete data are defined as missing values and missing variables (i.e. hidden or latent variables) in a dataset. Thus complete data are defined as a dataset where all cases have observations for all variables in the dataset.

Table 2-2 Classification of Missing Data

Classification	Missing-Data Mechanism
MCAR	independent of X and Y
MAR	Dependent on X and not Y
Systematic	Dependent on X and probably Y

Incomplete data can be classified into three categories based on the probability of observation, see Table 2-2. Given observed variables **X** and missing variable **Y**, if the probability of observation depends on **X** but not **Y** then the missing values are said to be missing at random (MAR). If the probability of observation is independent of both **X** and **Y**, then the missing values are missing completely at random (MCAR). Another way to look at MCAR is the missing values are MAR and the observed values are OAR (observed at random). If the probability of observation is dependent on both **X** and **Y**, the data are neither MAR nor MCAR, but missing systematically [Little and Rubin 1987]. Hidden variables are an example of data neither MAR nor MCAR. For this research,

with the exception of hidden variables, all missing values are assumed MAR. An example of MAR data is if in a survey, people are asked their occupation and salary. People in occupations that pay higher salaries may not wish to report their income. In this case, the missing values are dependent on occupations but not salary. Thus the data are missing at random.

Generally there are four approaches to dealing with incomplete data. One approach is to ignore cases with missing values. This approach may work well when data are MCAR and the number of cases is large. For this case the complete cases are a random subsample of the original sampled population. However, if the data are not MCAR, ignoring cases with missing values can lead to biased estimates. A more common approach in applied statistics is to impute (fill-in) the missing values with estimates. Special care must be taken with this approach to avoid biasing the estimates. Typical methods for imputing values are to use the mean values of the complete data or use a regression estimate of the complete data. A third approach is to estimate the expected statistics using weighted values. The weights are based on the estimated probability of the missing data mechanism. The final method is to estimate the values using model-based procedures. The most common of which is the expectation-maximization (EM) algorithm developed by Dempster, Laird, and Rubin [Dempster, Laird et al. 1977]. The EM algorithm begins with an initial random estimate of the parameter of interest. The next step (E-step) is to find the expectation of the parameter given the model. The expectation is then plugged into the model and the maximum likelihood is estimated. The parameter that gives the maximum likelihood (M-step) is then used to generate

another expectation, and so on. These iterations continue until the algorithm has converged, i.e. the parameters are no longer increasing [Little and Rubin 1987]. In their seminal paper, Dempster, Laird, and Rubin proved the EM algorithm is guaranteed to converge [Dempster, Laird et al. 1977].

2.5.1 Incomplete Data and Known Structure

What happens when the structure of the network is known, but the data used to learn the parameters are incomplete? This is certainly a more difficult problem than that of learning the parameters from complete data. The problem arises in how the missing values are handled. One approach would be to ignore all cases with missing values, but as mentioned earlier, this might lead to biases and inflated variance due to the smaller number of cases. This approach is only acceptable for a very small percentage of missing values. Another approach discussed previously is to use imputation where missing values are filled in using group means or regression, but these methods tend to lead to underestimates of the variability of the parameter estimates. For survey samples with missing data, Rubin has shown that multiple imputation works well. Multiple imputation is where the missing data are imputed multiple times to estimate the model [Rubin 1987].

A promising approach introduced by Lauritzen is to combine the EM algorithm with an inference algorithm [Lauritzen 1995]. The E-step of the EM algorithm is found by applying a Bayesian network inference algorithm. The estimates from the inference algorithm are then used in the M-step to maximize the likelihood. Lauritzen reports that "with data missing massively and systematically, the likelihood function has a number of

local maxima...." He also points out the slow convergence property of the EM algorithm and recommends an algorithm that uses the gradient.

Binder et al., took Lauritzen's advice and developed a parameter learning algorithm based on the gradient descent approach with a twist [Binder, Koller et al. 1997]. They did not learn parameters from data with missing values. They learned parameters from a known structure with hidden variables. The missing values in the data were those associated with the hidden variables. This is a slight twist to the problem tackled by Lauritzen, but since the data are missing systematically it is a much harder problem.

Missing Values	Difficult
Hidden Variables	Difficult
Missing Values and Hidden Variables	Most Difficult

Figure 2-11 Learning Structure from Incomplete Data

2.5.2 Incomplete Data and Unknown Structure

Arguably the hardest of the learning categories is inducing a structure from incomplete data. The problem can be broken down further into learning structure from data with missing values, learning structure with hidden variables, and learning structure with hidden variables and missing values, see Figure 2.11. This figure is an expansion of the right cell in Figure 2.9.

Why should one even care to learn Bayesian networks from incomplete data? The answer should be pretty obvious. Most situations we run into in real life contain missing data. It may be a medical database with missing symptoms, intelligent reports with

missing observations, malfunctioning sensor readings, or missing responses from surveys. In addition to missing values, one may be interested in networks with hidden variables. Friedman correctly points out that observed variables may only represent a fraction of the domain one is interested in modeling [Friedman 1998]. Take, for instance, a medical domain where only the symptoms and treatments are recorded. If the learner uses only the observable variables it will induce a network where all the symptoms and treatments are related. In fact, a more accurate model would include unobserved variables, e.g. diseases. If the patient's disease(s) is known, then the treatment is independent of most of the symptoms in the model. It is much easier to understand the model containing the disease variable than the model where most of the variables are related in ways that are unclear. Adding hidden variables may also simplify the model by reducing the number of distributions needed. Finally, in a more abstract way, inserting hidden variables in the network may help discover hypotheses. One could make the argument that many if not most of the concepts we use in everyday reasoning are "hidden variables" we have created to make sense out of our raw sensory inputs.

Very little work has taken place to learn network structures from incomplete data. Some of the first work was that of Cooper [Cooper 1995]. His approach was to insert a hidden variable in a database and enumerate all possible combinations of the hidden variable's values with each case in the original database. This approach effectively increases the number of cases in the database exponentially. Thus it is infeasible to learn networks with more than a few observable variables and a single hidden variable from moderate size databases.

Another early approach was that of Geiger et al, [Geiger, Heckerman et al. 1996].

They used Laplace's approximation and learned the maximum a posterior distribution.

The approach learned hidden variables as parents of the observable variables for small

networks. The major problem with this approach as point out by Friedman is that

Laplace's approximation assumes a unimodal likelihood function where in fact the

likelihood for this problem is characterized by multiple modes [Friedman 1998].

Singh introduced a technique that combines the EM algorithm with imputation

[Singh 1997]. His approach is to use the parameters from each step of the EM to derive

the conditional probabilities and then sample from the network to impute the missing

values. He creates several complete databases during each iteration and uses the

complete data algorithm, K2, to find the best structure from each dataset. He then merges

the structures using a heuristic based on arc frequency. The results showed that for this

approach multiple imputation (several samples of the missing data) performs better than a

single sample. The structure and parameters learned from the data with missing values

were relatively close to the structures and parameters learned using K2. A drawback to

this approach is that it requires prior knowledge of the correct node ordering to produce

the correct structure.

Arguably, the most significant advances in this area have been by Friedman, see

[Friedman 1998] and [Friedman 1998]. His approach, called Structural EM (SEM),

combines a greedy search over structures with the EM algorithm for estimating the

parameters. His approach begins with a given structure. Then it loops through the

following steps until it converges: 1) Computes the maximum a posterior (MAP)

parameters for the structure using EM. 2) Searches over structures for the highest score using the MAP from 1). 3) Let B_{n+1} be the highest scoring network from 2). 3) If $B_n = B_{n+1}$ then report convergence and return B_n, else go to 1).

Unlike Singh's algorithm, the SEM finds a structure with hidden variables from data with missing values. This is probably the most difficult learning problem discussed in this research. Though Friedman's results were encouraging, his algorithm and all the other algorithms described above, suffer the fate of any deterministic hill-climbing algorithm searching a multimodal landscape: it gets stuck in local optima. Friedman recommends multiple random restarts to overcome this problem.

2.6 Learning Framework Revisited

This chapter gave an overview of Bayesian networks and current research on learning Bayesian networks from data. From this information, several components of the learning framework object model can be instantiated for the problems addressed in this research.

- Phenomena: Stochastic observations with missing values.

- Prior Knowledge: Vague; number of variables in the network are known, but the structure is uncertain.

- Goal: Generalization, learn Bayesian networks.

- Constraints: Space and time.

- Assumptions: Missing data are MAR, all structures are equally likely, parameters are discrete.

- Representation: Bayesian Networks.

- Metric: see Chapter 3.

- Search: see Chapters 3 and 4.

Chapter 3 SEARCH METHODS FOR COMPLEX OPTIMIZATION AND LEARNING PROBLEMS

Most complex optimization and learning problems are characterized by a solution space that is very large, high dimensional, and multi-modal. The search space for learning Bayesian networks from incomplete data is no exception. Just considering the possible number of networks alone presents a daunting search space. Recall from (2.19), that the number of possible networks grows exponentially as the number of variables increases. Adding the problem of incomplete data causes the search space to become multi-modal. This has been noted by Lauritzen and Friedman [Lauritzen 1995], [Friedman 1998].

In the case of maximization, the search space for complex problems can be thought of as a landscape where the areas of good solutions are mountains and the areas of poor solutions are valleys. In fact, solution spaces are commonly referred to as landscapes in evolutionary algorithms. The function that gives rise to the landscape is commonly called the fitness function. A Bayesian network learning problem can be mapped into an optimization problem in which the objective function is the posterior probability of the Bayesian network.

A multi-modal landscape consists of many local maxima. These local maxima can be thought of as the peaks of the mountains. Local maxima are fixed points on the search landscape. A fixed point is said to be stable if given small perturbations, the trajectory of the search returns to that point. It is unstable if a small perturbation causes the trajectory to move away from the fixed point.

As mentioned in Chapter 1, there are two classifications of search algorithms for landscapes. A deterministic hill-climbing algorithm will always move to solutions with higher fitness. This deterministic move will always result in the algorithm finding a local maximum. Search stops when a local maximum is discovered. A stochastic algorithm on the other hand may accept, with probability, a move to lesser-fit territory. Stochastic algorithms are typically biased to find areas of higher fitness, but will not necessarily find a local maximum. If in fact they do hit a local maximum the search may continue as the trajectory moves back downhill in search of another local maximum.

This chapter addresses the final two instantiations of the learning framework for this research; search and performance metrics. Recall the first hypothesis put forth in Chapter 1 is that stochastic search approaches with populations will avoid the problem of getting "stuck" at local optima and generate "good" Bayesian networks from incomplete data. The stochastic search approaches I examined are evolutionary algorithms and Markov Chain Monte Carlo algorithms. The evolutionary algorithms are population based stochastic algorithms that take advantage of global information through use of the crossover operator. The Markov Chain Monte Carlo algorithms, when multiple chains are used, are population-based algorithms that use local information. In other words, the

next proposed state for a single chain in the MCMC is derived from the individual solution and no global information from the population is available.

3.1 Deterministic Algorithms

A typical deterministic hill-climbing algorithm searches a landscape always moving in the direction of increasing fitness (for this discussion I'll focus on finding the maximum value, the same concept applies to finding minimum values). The algorithm will look at solutions near its location and move to a solution with a greater value. At the maximum point of the hill it will stop the search because any neighboring solution will have a lower value than the current solution. Unfortunately, for complex solution spaces involving many modes, the "maximum" solution found by the deterministic algorithm could very well be a local maximum and may not even be in the set of what may be considered good solutions.

Take as an example the single dimension function with two modes given in Figure 3.1. Given the arbitrary starting position indicated, a deterministic hill-climbing algorithm will look at its two nearest neighbors and select the solution with the greatest value. As can be seen in the figure, when it reaches the top of the hill it can no longer move.

What is needed are search algorithms that allow the search to move downhill in the hope of finding another mode with even better solutions while maintaining a bias toward climbing hills. Stochastic search algorithms do just that. The remaining sections will discuss two such algorithms: evolutionary algorithms and Markov Chain Monte Carlo algorithms.
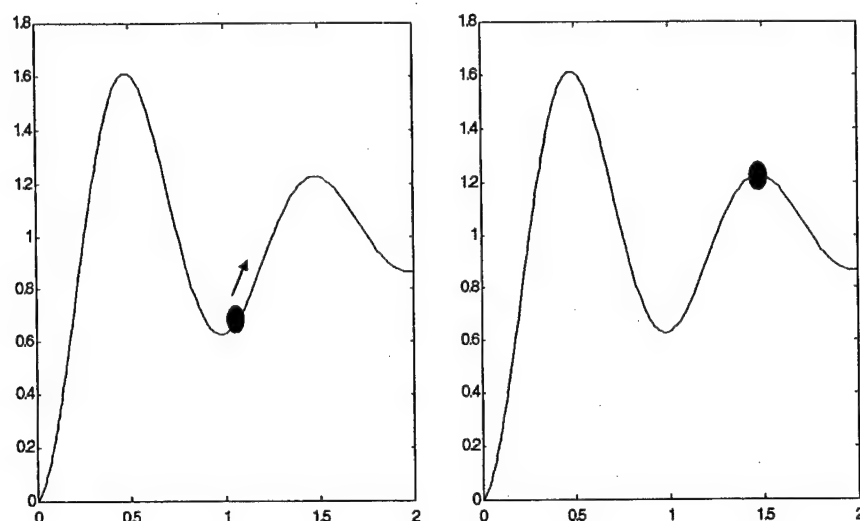
Figure 3-1 Two Mode Funciton a) search moves uphill b) search stops at local maximum

3.2 Evolutionary Algorithms

Evolutionary algorithms are a family of algorithms modeled after the organic evolutionary processes found in nature. They consist of a population of individual solutions that are selected and modified in order to discover overall better solutions in the search space. More specifically, the algorithms proceed as follows. An initial population of solutions is generated. Until some stopping criterion is met the population is evolved in the following manner. Individuals are selected from the population based on fitness. The fitness of an individual is determined by how good a solution it provides. For example, if the problem is to find the maximum value of a function, the individuals may represent some coordinate values and the fitness of the individual is the value of the function given the individual. The selected individuals are then modified using genetic operators. The most common genetic operators are crossover and mutation. In

crossover, two parent individuals are selected and information is exchanged between the individuals at selected points, see Figure 3.4. In mutation, a single individual is modified in some way, such as flipping a bit in a binary string representation. The resulting individuals are known as the offspring. The final step of the evolutionary algorithm is to select the next generation from the current parent population and the offspring population. This new generation goes through the process again unless the stopping criterion is met.

Bäck formally describes an evolutionary algorithm as an 8-tuple, presented here slightly modified,

$$EA = \{I, \Phi, s, \Omega, \mu, \lambda, \iota, \Psi\} \tag{3.1}$$

where I represents the individual, Φ is the fitness function, s is the selection operator, Ω is the set of genetic operators, μ is the number of individuals in the parent generation, λ is the number of individuals in the child generation, ι is a terminator function, and Ψ is the process of transforming a current population to the next population by applying the genetic operators and selection [Back 1996]. The sections below will discuss in some detail issues of representation $\{I\}$, selection $\{s, \mu, \lambda, \Psi\}$, and operators $\{\Omega, \Psi\}$. The fitness function $\{\Phi\}$ is problem dependent. For learning Bayesian networks, the Bayesian Dirichlet score (2.21) is the most common fitness function (i.e. performance metric) and is used exclusively in this research.

Evolutionary algorithms are the products of three independent and parallel developments from the 1960s: evolution strategies, evolutionary programming, and

genetic algorithms. Evolution strategies were developed in the early 1960s by Bienert, Rechenberg, and Schwefel to tackle very complex multi-parameter hydrodynamic engineering problems. These problems were too difficult for standard numerical optimization algorithms. Their approach was to generate a population of μ individuals representing potential solutions. The μ parent individuals are then copied and mutated, and in some cases a form of recombination from a set of parents is used, to create λ offspring, with $\lambda > \mu$. The parents and children are then culled down to μ individuals for the next generation. This is commonly done in two ways. One approach is to combine both parents and children, sort them, and choose the best μ individuals. This is known as $(\mu+\lambda)$ evolution strategy. The other approach ignores the parent generation and selects the best μ individuals from the child population. This strategy is known as the (μ,λ) evolution strategy. Evolution strategies are typically used when confronted with a multi-parameter real-valued optimization problem [Schwefel 1995].

Evolutionary programming also came about in the 1960s. L. Fogel developed this algorithm as a means of evolving finite state machines and his son, D. Fogel extended his work to continuous value problems. Evolutionary programming is very similar to evolution strategies. Like evolution strategies, evolutionary programming evolves a population of potential solutions. With evolutionary programming, the only genetic operator used is mutation. With the canonical algorithm both the parent and child populations are combined and the next population is generated by stochastic selection based on fitness. Typically, selection is performed by randomly selecting two individuals

from the combined pool and selecting the one with best fitness value to survive [Fogel 1991].

At the same time evolution strategies and evolutionary programming were being developed, Holland was independently working on a different evolutionary approach called genetic algorithms. Holland's approach was to map the potential solution, called the phenotype, into a common representation known as the genotype that could then be operated on by a common algorithm. In other words, despite the domain, the genotype representation would always be the same; therefore, the algorithm evolving the solutions could stay the same. The representation for the common genotype Holland chose was a binary bit string. His belief was that any problem could be reduced to this representation. As in the other evolutionary approaches, genetic algorithms evolve a population of solutions. Unlike the other two, the main genetic operator is crossover of information from two selected individuals. The selection process for genetic algorithms is to choose parents for the offspring population with a stochastic selection based on fitness. For the canonical genetic algorithms all parents are then killed off and only the offspring survive [Holland 1995].

Over the past few years, the line distinguishing the three approaches has begun to blur as researchers blend the concepts from each of the above algorithms into recipes for solving complex problems. Since they are very much related, it's easy to see how one can pick and choose the best components from each and combine them into an algorithm that works well for a specific domain. Thus the name evolutionary algorithm was coined to capture any algorithm meeting the traits defined in (3.1) above.

The work presented herein has a distinct genetic algorithm flavor. Many of the concepts and theory discussed are from that branch of evolutionary algorithms. That's not to say that an algorithm that more resembles evolution strategy or evolutionary programming may not work just as well or better. This is the subject of further research.

To many, the idea of evolutionary algorithms brings to mind a family of algorithms for optimization. In fact, evolutionary algorithms are not necessarily good at finding optimal solutions. For complex problems, the global optimal solution may be very difficult to locate for any algorithm. This leads to a trade-off between exploring the landscape for areas that appear to hold good solutions and exploiting the good areas found. This can be a very difficult balancing act. Too much exploration may result in not finding the local optima in the regions explored, while too much exploitation can lead to behavior similar to a greedy hill-climbing algorithm.

Perhaps a better way to think of evolutionary algorithms is as a search strategy for an emergent behavior [DeJong 1993]. Through a balance of exploration and exploitation, the emergent behavior is a set of overall better solutions. Holland suggests that we should view genetic algorithms as optimizing a sequence of decision processes given a limited number of trials in order to maximize cumulative payoff in the face of uncertainty [Holland 1995]. The balance of exploration and exploitation is accomplished by optimizing population size, genetic representation, and genetic operators for the problem domain. This in and of itself can be a very difficult problem. In fact, there has been a significant amount of theoretical work trying to find models for optimizing these parameters. To date, no silver bullet has been found. This leaves us with empirical

analysis to find a set of parameters that work well for each domain. The research herein is no different.

The next few sections will look at a few of the evolutionary parameters. The first section is a brief discussion of representation. Some evolutionary algorithm terminology will be given and the representation (i.e. internal representation) used in this research described. The next two sections describe selection and genetic operators. This is the area where much of the empirical assessment will be conducted. Population size, although an important parameter, will not be discussed per se. The population size for these empirical analyses described in this research will be limited because of the time complexity required to score a Bayesian network, especially a network of moderate size.

3.2.1 Representation

One of the more critical design decisions required when developing evolutionary algorithms is that of representation. In evolutionary algorithms there are two types of representations. When a representation is stated in its natural state, in the problem domain, it is said to be a phenotype. For example, a real value optimization problem would be represented as a set of real valued coordinates. On the other hand, it may be necessary to map a phenotype representation into another structure to make it easier for the algorithm to modify and exchange information. In many cases the phenotype can be mapped into a representation that resembles, at a very high level, a sequence of local structures or building blocks. In this case, the representation is referred to as a genotype.

This is a good place to quickly cover some terminology commonly used in evolutionary algorithms. As should be expected many of the terms used in evolutionary

algorithms are common to natural evolution. For example the term phenotype in natural evolution refers to the expressed characteristics of an individual. In evolutionary algorithms the term has the same meaning. The phenotype is what you observe in the domain. The genotype, on the other hand, in natural evolution is the genetic material inherited from the individual's parent or parents. In artificial evolutionary algorithms a genotype represents a mapping from the phenotype to "genetic material" like structure. The genotype is composed of chromosomes. In most evolutionary algorithms there is a single chromosome. Each chromosome is composed of a string of genes. The genes can be thought of as a substructure or feature of the phenotype. For binary bit strings the genes may represent a hyperplane in a real valued coordinate space. For Bayesian networks the genes may represent a substructure of a directed acyclic graph such as a variable and its parents. An allele is the actual value of a particular gene. The position of a gene in a chromosome is known as the locus [Goldberg 1989].

Learning Bayesian networks from incomplete data involves learning or inferring three components: the structure and parameters of the network and the missing data. This can be simplified to learning the structure of the network and the missing data, since if one learns the structure the parameters can be found with ease. Under Bayesian Dirichlet scoring if the structure and data are known, parameter estimates can be derived in closed form. One way to learn the structure and missing data is to evolve them. The approach taken here is to evolve structure and missing values simultaneously. This amounts to evolving two parallel but dependent populations. Thus two different representations are needed. Each will be represented genotypically as chromosomes.

Let the data be the matrix **D** where the columns are the variables **V** and rows are the individual cases **C**. Each case C_i, is by assumption, an independent and identically distributed (iid) random variable. Missing values are always assumed to be MAR except for the case of hidden variables in which case data are missing systematically. Let **D** = {**X**,**Y**}, where **X** represents the observed values and **Y** represents the missing values. The vector **Y** can be represented as a chromosome with genes representing each missing value y_i and alleles taking the values of y_i, see Figure 3.2.
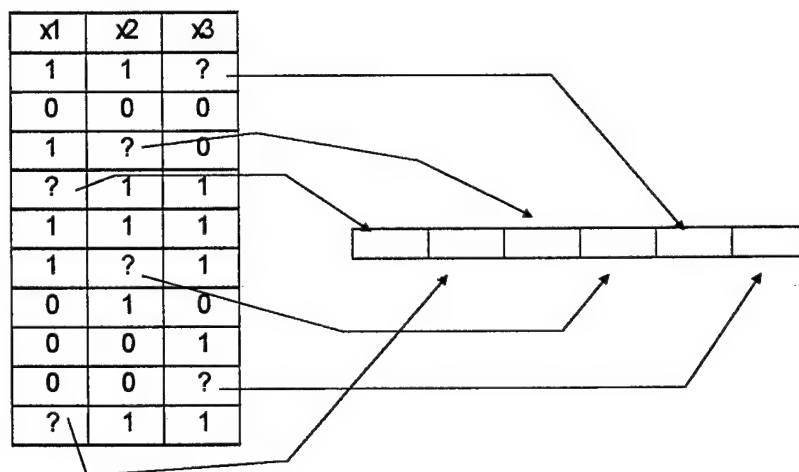
Figure 3-2 Missing Data Chromosome

The structure B_S can be represented as an adjacency list, see Figure 3.3, where each row represents a variable V_i and the members of each row, with the exception of the first member, are the parents of V_i, pa(V_i). For clarity purposes only, the first member of each row of Figure 3.3 shows the value of the variable V_i. The adjacency list can be thought of as a chromosome, where each row is a gene and the pa(V_i) are the alleles. This representation is convenient because from equation (2.21), the logarithm of the scoring

metric is the summation of scores for each variable. Because of this, each gene can be scored separately and added to generate the fitness score for the entire structure.
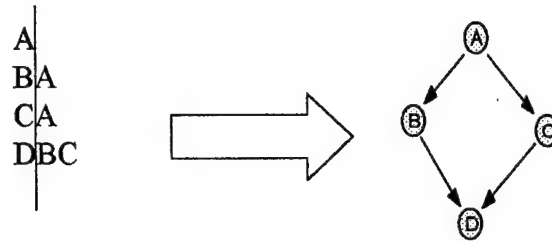
Figure 3-3 Structure Chromosome

The allele values that each gene can take on can become enormous. The values can range from no parents to n-1 parents where n is the number of variables in the data set. Thus an allele can take on $\sum_{i=1}^{m}\binom{n-1}{i}$ possible values where m is the maximum set of parents a variable can have and n is the number of variables in the data set. Because of the exponential growth of computation time, it is wise to limit the number of parents for a variable. As an example of the large size of allele values take n=11 and m=4, the number of possible values for a given allele is 376, while if n=41 and m=4, the number of possible values grows to 102,091.

In addition to the large combination of allele values per gene, the genes are highly correlated. This is because the values of the genes (i.e. the parents of the variable), if not empty, are also genes. For example, the allele value of gene B in Figure 3.3 is A and when mapped to a phenotype it means that A is a parent of B. Variable A is also a gene in Figure 3.3. Many combinations of genes can lead to illegal structures; in other words, structures that are not directed acyclic graphs. This problem is alleviated by arbitrarily

assigning illegal structures a very low score. In essence, this amounts to assigning a very low non-zero probability to illegal structures. The reason for allowing illegal structures is the chromosome may contain very good genes that can be reconstituted as building blocks for even better structures through recombination or mutation.

3.2.2 Selection

3.2.2.1 Introduction

Selection in evolutionary algorithms is the process of choosing which individuals reproduce offspring and which individuals survive to the next generation. When selection is used to choose which individuals reproduce, the process is referred to as pre-selection. When it is used to select the individuals that survive to the next generation it is called post-selection. Selection can further be categorized as deterministic or probabilistic. Deterministic selection tends to behave more like greedy hill-climbing algorithms and exploit the nearest areas with promising solutions. Probabilistic selection schemes are more exploratory and search the landscape.

An important decision required when deciding on what type of selection schemes to use are whether to emphasize exploration or exploitation. When a scheme is more exploratory it is said to have a low selection pressure. When a scheme is more exploratory it is said to have greater selection pressure. In other words, selection pressure is a vague measure of how often more fit individuals are selected to reproduce and/or live to the next generation. The key is to find a selection pressure that balances exploration and exploitation.

Selection pressure can be measured by how well it selects individuals with a higher fitness. Goldberg defined the *birth, life, and death equation* as

$$\mu_{i,t+1} = \mu_{i,t} + \lambda_i - \mu_{i,d} \qquad (3.2)$$

where μ_i is the number of individuals of type i in the population, $\mu_{i,t}$ is the number of individuals of type i at time t, $\mu_{i,t+1}$ is the number of individuals of type i at time t+1, λ_i is the number of individuals of type i born at time t, and $\mu_{i,d}$ represents deaths of individuals of type i from the current population [Goldberg and Deb 1991]. Perhaps a more appropriate way to look at the *birth, life, and death equation* is in terms of the expected proportion

$$< \mu_{i,t+1} > = < \mu_{i,t} > + < \lambda_i > - < \mu_{i,d} > \qquad (3.3)$$

where $\langle \cdot \rangle$ represents the expected value.

A common metric for characterizing selection pressure is to find the takeover time for the particular selection scheme [Goldberg and Deb 1991]. The takeover time is a measure of how many generations it takes for the best individual in a population to take over the entire population when evolution is through selection only with no application of genetic operators.

Selection schemes can be further categorized into generational or steady-state schemes. A selection scheme is generational when the entire current population is replaced by its offspring to create the next generation. For generational selection schemes, equation (3.3) simplifies to $<\mu_{i,t+1}>=<\lambda_i>$ since $<\mu_{i,t}> - <\mu_{i,d}>=0$. A scheme is referred to as steady-state when a select few offspring replace a few members of the

current generation to form the next generation. Another set of categorizations along these lines are (μ,λ) and (μ+λ) defined above for evolution strategies. The (μ,λ) classification is equivalent to generational. The (μ+λ) schemes are somewhere between generational and steady-state since none, some, or all of the offspring replace the current population.

The selection schemes in this research can be categorized as probabilistic pre-selection with deterministic post-selection. The deterministic post-selection is simply a deterministic decision to replace the current population with its offspring (generational model). To increase selection pressure, the algorithms considered herein also employ an elitist strategy. This is where the highest fit individual from the current population is always carried over to the next generation. This guarantees that the highest fit individual is not lost from generation to generation and is available to pass on its genetic information to each generation through crossover and/or mutation. The probabilistic selection schemes considered are fitness proportional, rank proportional, and binary tournament. Each of these is described below and an empirical analysis is given in chapter 5.

3.2.2.2 Fitness proportional

The most common selection scheme is to probabilistically select individuals based on their fitness in proportion to the other individuals in the population. This is the selection scheme originally proposed by Holland and used in canonical genetic algorithms [Holland 1995]. For fitness proportional selection, individuals of type i are

selected with probability $p_{i,t} = \dfrac{\Phi_i}{\sum\limits_{j=1}^{k} \mu_j \Phi_j}$ where k is the number of different types of

classes of individuals, Φ_i is the fitness of individual i, and the total number of individuals

sum to n. From (3.3) the expected number of type i individuals is $<\mu_{i,t+1}> = <\mu_{i,t}> \dfrac{\Phi_i}{\overline{\Phi}_t}$,

where $\overline{\Phi}_t$ is the average fitness of the population at time t.

Fitness proportional selection has worked very well for many problems. It will

quickly locate areas in the landscape with good solutions. Goldberg and Deb found the

takeover time for proportional selection to be on the order $O(\mu\ln\mu)$ [Goldberg and Deb

1991]. In other words, in the absence of genetic operators, the best individual from a

population of $\mu=20$ will takeover the population in about 60 generations. However,

DeJong showed that it lacks a killer instinct [DeJong 1975]. There are essentially two

phases of the search process. When the fitness values are widely scattered, as typically

occurs in the early search when genetic material is diverse, the search proceeds very

quickly. However, when the difference in fitness values become small, as typical of the

time when the genetic material has begun to converge, the selection pressure decreases

significantly and fitness proportional selection approaches random selection. To counter

this, DeJong proposed scaling the fitness values. While this approach shows some

improvement, there is no theory or overall empirical results that tell us the best scaling

functions to use and when. This is still the subject of active research.

3.2.2.3 Rank proportional

One way to increase selection pressure is to rank the individuals in the population by fitness and select for reproduction based on a probability proportional to rank. Baker was the first to introduce this scheme and identified the following three advantages [Baker 1985]. First, rank selection avoids the problem of determining an appropriate scaling function. Next, rank selection can be used to control the selection pressure more directly than fitness proportional selection. Finally, there is a significant speed up in search. The most common ranking function is linear, but nonlinear functions can also be used.

In Baker's original linear ranking function, he parameterized the maximal expected value to control the slope of the linear function. Let the individuals be ranked by fitness values such that, given μ individuals, the best fit individual is indexed as $\vec{a}_1$, the next best as $\vec{a}_2$, and so on with the worst fit individual is $\vec{a}_\mu$. The maximal expected value, η^+ is assigned to $\vec{a}_1$ and the minimal expected value, η^- is assigned to $\vec{a}_\mu$. The linear mapping to the probability of selection of an individual of type i is

$$ p_i = \frac{1}{\mu}\left(\eta^+ - (\eta^+ - \eta^-)\cdot\frac{i-1}{\mu-1} \right) \tag{3.4} $$

Since p_i is a probability, then $\sum_{i=1}^{\mu} p_i = 1$ and $p_i \geq 0 \ \ \forall i \in \{1,\ldots,\mu\}$, thus $1 \leq \eta^+ \leq 2$ and $\eta^- = 2 - \eta^+$ [Back 1996]. The takeover time for linear rank selection schemes are of the order O(lnμ), a factor of μ faster than fitness proportional selection.

3.2.2.4 Binary Tournament

The final selection scheme considered in this research is binary tournament selection. In this approach, two individuals are randomly selected from the population and compared. The one with the highest fitness is selected for reproduction. Then another two individuals are randomly selected and the best fit kept as the mate to the first parent. This process continues μ times until the next generation is populated. Tournament selection can be generalized from binary tournament selection to k-ary tournament selection.

The probability of selecting an individual i for k-ary tournament selection is given by

$$p_i = \frac{1}{\mu^k}((\mu - i + 1)^k - (\mu - i)^k) \tag{3.5}$$

where individuals have index i based on fitness as in rank selection above. The takeover time for tournament selection was shown by Goldberg and Deb to also be of the order of $O(\ln\mu)$ [Goldberg and Deb 1991]. All things considered equal, tournament selection should be preferred over rank selection since it is the easiest of the three to implement: randomly select k individuals and pick the best versus converting the fitness of individuals to a function and drawing from the function.

3.2.3 Operators

3.2.3.1 Mutation

Mutation is a genetic operator by which small modifications are made to a single genotype or phenotype of a selected individual. For real-valued phenotype approaches such as those found in evolution strategies or evolutionary programming, the mutation operator is usually of the form of a small Gaussian change to the phenotype. For binary coded genotypes typically found in canonical genetic algorithms a mutation is a random bit flip. Yet for more complex genotypes and phenotypes, mutation can take on many varied forms. Typically though, it is a minor random adjustment to the selected individual.

Mutation is applied to an individual with probability, p_m. Evolution strategies and evolutionary programming rely heavily on mutation. For evolution strategies p_m is typically close to one and for evolutionary programming $p_m=1$. However, in genetic algorithms, mutation is considered a background operator and p_m is usually given a very low probability in the range of 0.001 to 0.01. For domain dependent evolutionary algorithms the optimal mutation rate may be low or high and in many cases must be determined through empirical analysis.

The reason for applying mutation is that a randomly generated initial population may not contain the alleles necessary to find the global maximum. In fact, it may not be possible to find even a "good" solution with the available alleles. Mutation is the only operator that modifies values of genes. Therefore it is the only operator that creates new alleles in the population.

Most mutation operators are fairly straightforward. For the missing data chromosome, each gene in an individual is selected for mutation with probability p_m which is usually a very low probability, [0.0001 0.001]. The mutation operator for the missing data chromosome proceeds as follows. A gene is selected and the current value of the gene is replaced probabilistically by another value from the value set of the variable corresponding to that gene. In other words, the selected gene maps to variable X. The gene contains the value 4. If the variable X has possible values X = {1,2,3,4,5}, the genes current value is replaced by a random draw from the set {1,2,3,5}. The length of the missing data chromosome depends on the amount of missing data in the data set so it can be very large. For example for a dataset with 1000 cases, 10 variable, and five percent missing data, the missing data chromosome has 500 genes. If a hidden variable is added to the above example the length increases to 1500 genes. Also, the allele values depend on the possible number of values that the corresponding variable can take on. This results in a very large combination of possible genes.

The mutation operator for the structure chromosome is tailored to the representation we used and its mapping to a directed graph phenotype. Recall the gene of the structure chromosome represents the gene's parent nodes in the graph. We include two basic modifications to a gene: add a node and delete a node. These operators have the effect in the phenotype of adding and deleting arcs, respectively. We also include a third basic modification, reversal of an arc, which is implemented genotypically by deleting the parent-child arc and adding the child-parent arc.

3.2.3.2 crossover

Crossover is where two or more individuals exchange information to create one or more new individuals. The use of crossover as the main genetic operator is one of the main separating characteristics between genetic algorithms and other evolutionary algorithms. In its simplest form, as introduced by Holland, a single random locus is selected as a slice point and the segments are exchanged between two individuals at the locus, see Figure 3.4. This is called single-point crossover. Multi-point crossover is when more than one locus is selected and information exchanged in segments between the locus points. Another form of crossover is when each gene is, with probability, swapped between two individuals. This scheme is known as parameterized uniform crossover.
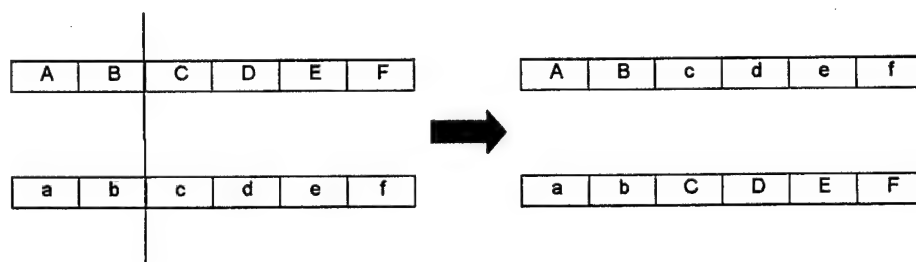
Figure 3-4 Crossover

Traditional theory introduced by Holland and explored empirically by Dejong expounded on the virtues of using 1- or 2-point crossover [Holland 1995], [DeJong 1975]. The reason for this is that fewer crossover points cause less disruption between two or more good performing alleles. This will be discussed further in section 3.2.4. Yet empirical results suggests that n-point and uniform crossover can perform better than 1-

or 2-point crossover for some problems [Eschelman, Caruana et al. 1989], [Syswerda 1989]. DeJong and Spears found there are two main reasons for multi-point and uniform crossover outperforming 1- or 2-point crossover in these studies. First, as the populations become more homogenous disruption is required to produce new individuals. Since a higher number of crossover points produce more disruption than a lower number of points, it would produce new individuals more often continuing exploration of the landscape. The second reason found for the performance difference was an interaction between population size and number of crossover points. Smaller populations performed better with larger number of crossover points. This is because the smaller populations become homogenous more quickly than larger populations since the smaller size has less information capacity than a larger population [DeJong and Spears 1990].

Because of the time complexity involved in computing the fitness function for Bayesian network structures, the population size is limited. With this in mind, the crossover operator of choice for both the missing data chromosome and the structure chromosome is parameterized uniform crossover. Crossover for the missing data chromosome looks no different than parameterized uniform crossover of a bit string. The crossover operator for the structure chromosome is shown in Figure 3.5 along with the resulting phenotype structures. Notice the similar substructures in the phenotype between the parents to the offspring.

3.2.4 Theory

There has been a significant amount of empirical evidence that evolutionary algorithms work very well for many problem domains. In addition, there has been a

significant amount of theoretical work on canonical algorithms. Unfortunately, the theory covers only the simple versions of the algorithms and it's not always clear if the theory of simple evolutionary algorithms extends to more complex versions of these algorithms. Because of the lack of theoretical proofs of performance, it's very difficult to tell, without prior knowledge, if an evolutionary algorithm is converging to areas in the search space with "good" solutions prior to completion of the search. Clearly, more theoretical work is required and this is an area of active research. The following sections will very briefly touch on some of the more common theoretical foundations for evolutionary algorithms. Because the work herein has more of a genetic algorithm flavor the convergence theorems for evolution strategies and evolutionary programming will be skipped.
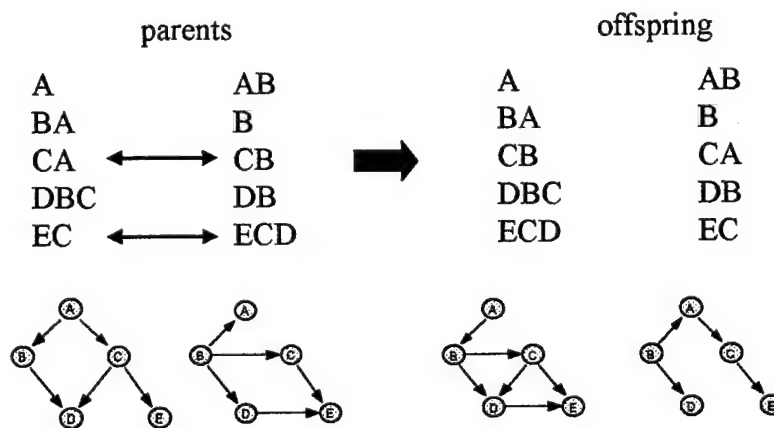
Figure 3-5 Uniform Crossover of Network Structures

3.2.4.1 Schemata Theorem

Holland laid the original theoretical groundwork for genetic algorithms. His approach was to show that genetic algorithms worked by allocating more samples to good building blocks and recombining them into better solutions. The building blocks are the genes that contribute higher fitness to the individuals in which they are present. Holland used the term schema instead of building blocks and referred to his theory as the schemata theorem.

The schemata theorem, as defined by Holland, applies to the simple genetic algorithm with fitness proportional selection, a very low mutation rate, and 1-point crossover. A schema is a bit string template made up of the set {0,1,*}, where * represents a "don't care." For example the schema $H = 1**0$ represents all sets of 4-bit strings with a leading 1 and a trailing 0. It consists of the set {1000, 1010, 1100, 1110}. The *order* of schema H, o(H), is the number of fixed positions in the schema template. The order of the schema above is 2 because the leading 1 and trailing 0 are fixed. The *defining length*, δ(H), is the distance between the first fixed position in the schema and the last fixed position. The defining length for the example above is 2 since there are two bit positions between the first and last fixed position. The schema theorem is thus defined as

$$\langle \mu(H,t+1) \rangle \geq \langle \mu(H,t) \rangle \frac{\Phi(H)}{\overline{\Phi}} \left[1 - p_c \frac{\delta(H)}{l-1} - o(H) p_m \right] \qquad (3.6)$$

where $\langle \mu(H,t+1) \rangle$ is the expected number of schema H in generation t+1, $\langle \mu(H,t) \rangle$ is the expected number of schema H in generation t, $\Phi(H)$ is the average fitness of schema

H in the current population, $\overline{\Phi}$ is the average fitness of the population, p_c is the probability of crossover, l is the length of the chromosome, and p_m is the probability of mutation. For a complete proof of the schema theorem see [Holland 1995] or [Goldberg 1989].

The schema theorem gives a lower bound on the expected number of a particular schema from generation to generation. Notice that it implicitly predicts the expected number of a particular schema from an explicit calculation of population fitness and an implicit calculation of schemata fitness. Holland referred to this as implicit parallelism. From (3.6), one can see that more fit schema will be allocated an increasing number of samples while lesser-fit schema will be allocated fewer samples. In addition, schemata with shorter defining lengths are more likely to survive than schemata with longer lengths.

3.2.4.2 Markov Chains

A Markov chain is a memoryless process where only the current state of the process influences where the process will go next. If we think of a state of a genetic algorithm as distribution of individuals in the current population, then from (3.6), it's easy to see that the next state of the genetic algorithm is influenced only by the current state. There have been several studies of genetic algorithm behavior using Markov chains. Each of these studies has concentrated on simple genetic algorithms with fitness proportional selection, binary bit strings, 1-point crossover, and mutation, see [Davis and Principe 1991], [Vose 1993], [Nix and Vose 1992], [Suzuki 1993], [Rudolph 1994], and [DeJong, Spears et al. 1995].

One particularly interesting study is that of Nix and Vose, [Nix and Vose 1992]. They were able to derive the exact transition probabilities of a simple genetic algorithm with a finite population. The transition probabilities are defined in terms of a transition matrix, $\mathbf{Q}$, where $\mathbf{Q}_{i,j}$ is the probability of moving from state i to state j. In order to define $\mathbf{Q}$, Nix and Vose first found a matrix $\mathbf{Z}$ that enumerates all possible configurations of populations.

Let the length of the chromosome be l, the total number of possible chromosomes represented as binary bit strings is $r=2^l$. If the population size is n, then the total possible number of unique populations, N is

$$N = \binom{n+r-1}{r-1}.$$ (3.7)

For a chromosome of length 2 and population size of 2 the number of unique population configurations is N=10. For purposes of enumeration, each possible chromosome configuration is mapped and indexed with a unique integer value from <0,..,r-1>. The $\mathbf{Z}$ matrix is an Nxr matrix that enumerates all possible populations where the ith row, defined as $\phi_i = <z_{i,0},...,z_{i,r-1}>$, represents the vector for the ith population. The cells of the $\mathbf{Z}$ matrix contain the number of occurrences of each chromosome in each population. For example, the cell $z_{j,y}$ represents the number of occurrences of the chromosome indexed as j for the population indexed as y.

In order to complete the equation for $\mathbf{Q}$, Nix and Vose defined the operators F and M, where F is a function of the fitness function and M is a function of the genetic operators: mutation and crossover.

$$Q_{i,j} = n! \prod_{y=0}^{r-1} \frac{\left[M\left[\frac{F\phi_i}{|F\phi_i|} \right]_y \right]^{z_{j,y}}}{z_{j,y}!} \qquad (3.8)$$

where $|\vec{w}|$ is the summation of entries in vector $\vec{w}$, and $[\vec{w}]_y$ is the yth component of $\vec{w}$. For a full proof and complete definitions of F and M see [Nix and Vose 1992].

All the entries of Q are non-zero when mutation is non-zero. Hence all states of the Markov chain can be reached from any other state. This means the Markov chain is ergodic. It is well known [Feller 1968] that any ergodic Markov chain converges to a unique steady state distribution. This implies that the probability of reaching any state, i.e. any genetic algorithm population composition, does not depend on the initial state of the chain.

Another implication of the Markov chain of a genetic algorithm being ergodic, is the genetic algorithm will escape all local maxima (Vose used local minima). Vose showed, [Vose 1993], that as the Markov chain converges to its steady-state distribution, the genetic algorithm will concentrate on sampling points near local maxima. He also shows that these points are not equally likely and the genetic algorithm in fact will asymptotically converge to the local maximum with the largest basin of attraction. This, of course, is not necessarily the global maximum thus proving DeJong's conjecture that genetic algorithms are not function optimizers [DeJong 1993].

Equation (3.8) defines a model that can be used to compute the trajectories of a simple genetic algorithm. Unfortunately, for real-world problems of even relatively small size, the transition matrix is enormous making computation impossible. For

example, if $l=10$ and $n=20$ the number of unique populations required for the **Z** matrix grows to $N \approx 8 \times 10^{41}$. However, as Vose demonstrated, some very useful properties can be derived from the model.

3.2.5 An Evolutionary Algorithm for Learning Bayesian Networks from Incomplete Data

As mentioned in chapter 2, one of the main problems with current approaches to learning Bayesian networks from incomplete data is that the search is deterministic and the landscape for the problem is very large, multi-dimensional, and multi-modal. In order to avoid climbing the nearest mode, deterministic algorithms must be rerun with random starting points. Finding the global maximum for this size of problem is like searching for a needle in a haystack. However, we can attempt to find areas containing good solutions. One hypothesis from this research is that evolutionary algorithms will find such areas.

Finding the best search schemes, genetic operators, and parametric settings for an evolutionary algorithm still requires empirical analysis. The algorithm provided below allows parameters to be modified to find the right mix of selection, crossover rates, and mutation rates. Other parameters can also be modified with great ease. There are three search schemes used and assessed in the algorithm below: fitness proportional, rank proportional, and binary tournament. The genetic operators are crossover and mutation. Since the population size is small, by necessity, the crossover operator used is parameterized uniform crossover as suggested by DeJong and Spears [DeJong and Spears 1990].

The algorithm is unique in that there are two populations evolved simultaneously: the missing data and network structure. The idea is that the missing data and structure will evolve in parallel to high scoring networks. The algorithm is given in Figure 3.6.
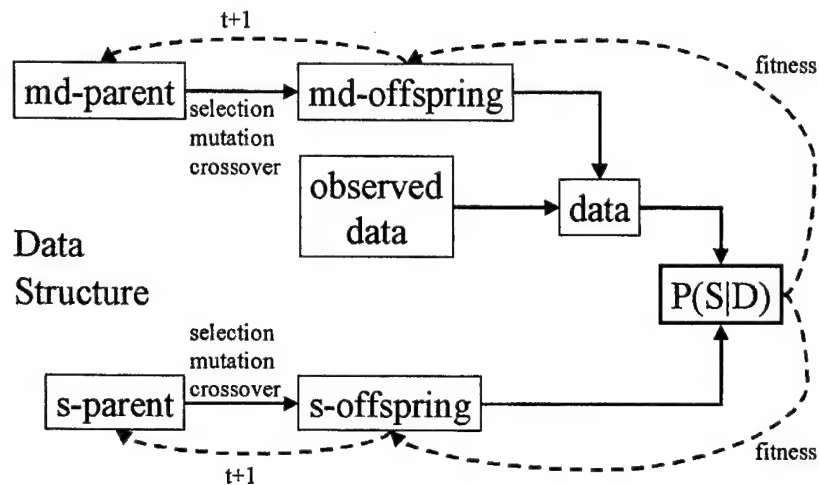
Figure 3-6 Evolutionary Algorithm for BN from Incomplete Data

3.3 Markov Chain Monte Carlo Algorithms

3.3.1 Introduction

Markov Chain Monte Carlo is a general class of algorithms used for optimization, search, and learning. These algorithms arose in statistical physics where they were used as models of physical systems that seek a state of minimal free energy. These algorithms have also been used more recently in statistical inference and artificial intelligence [Neal 1993], [Gilks, Richardson et al. 1996], [Geman and Geman 1984].

I'll begin with a brief introduction to terms from statistical physics and then relate them to probability theory and search in a fitness landscape. Physical systems can be

described in terms of their macrostates and microstates. A macrostate is a system's observable components, such as temperature or pressure. A microstate is the non-observable detailed state of the system's atomic components, such as position and velocity of every molecule in the system. One goal of statistical physics is to model a system's microstate from information about its observable macrostate.

Each microstate has an associated energy, E(s). An isolated system free of external influences is assumed to evolve to an equilibrium state that minimizes a quantity known as *free energy*. This equilibrium is probabilistic—that is, it is not possible to know the microstate in detail, but only to predict the probabilities of various microstates. The equilibrium probability that a system is in a given microstate is

$$P(s) = \frac{1}{Z} \exp(-E(s)/T) \tag{3.9}$$

where Z is a normalization constant which ensures that the probabilities sum to 1, and T is the temperature. This distribution is commonly known as the Boltzman distribution. As will be seen later, this distribution is important in Markov Chain Monte Carlo algorithms. The Boltzman distribution (3.9) is the probability distribution over microstates that minimizes the free energy, subject to a constraint on the expected value of the total energy of the system. The free energy is defined as $F = <E> - TS$, where F is free energy, <E> is the expected value of the energy, T is temperature, and $S = -\sum_{\tilde{s}} P(\tilde{s}) \log(P(\tilde{s}))$ is the entropy. MCMC algorithms for optimization or statistical inference involve creating a mapping from the minimal free energy distribution (3.9) to the solution of the optimization or inference problem.

Statisticians recognized that any probabilistic inference problem could be translated into the language of statistical physics. The probability distribution of any random variable can be represented as a Boltzman distribution over the microstates of an imaginary physical system. If S is the sample space and P(s) is the probability of occurrence of $s \in S$, then we define an associated energy as:

$$E(s) = -T[\log P(s) + Z] \tag{3.10}$$

where T>0 is an arbitrary constant and Z is chosen as the solution of the equation

$$Z = \sum \exp\{-T[\log P(s) + Z]\} \tag{3.11}$$

It is clear that P(s) is the Boltzman distribution of the system with energy values E(s) and temperature T. Note that when T=1, E(s)=-logP(s).

Consider a generic problem of inferring the posterior distribution of a parameter θ from a set of observations $x_1,...,x_C$, where the observations are assumed to be independent samples from the distribution $P(x|\theta)$. We can define an energy function with T=1 as the joint probability of the observations and the prior distribution:

$$E_C(\theta) = -\log(P(\theta)P(x_1,...,x_C|\theta)) = -\log P(\theta) \prod_{i=1}^{C} P(x_i|\theta)) \tag{3.12}$$

The normalization constant for the posterior distribution is

$$Z_C = \int \exp(-E_C(\tilde{\theta}))d\tilde{\theta} = P(x_1,...,x_C) \tag{3.13}$$

So the posterior distribution is

$$P(\theta|x_1,...,x_C) = \frac{P(\theta)\prod_{i=1}^{C} P(x_i|\theta)}{P(x_1,...,x_C)} = \frac{1}{Z_C}\exp(-E(\theta)) \tag{3.14}$$

exactly the Boltzman distribution [Neal 1993]. In the language of statistical physics, then, conditioning is accomplished by setting the energy equal to the logarithm of the joint probability of data and parameters, setting the temperature to 1, and normalizing over the parameter while holding the data fixed at their observed values.

The representation of (3.13) for a general statistical inference problem means that algorithms from statistical physics for estimating minimum free energy configurations of physical systems can be applied to general problems in statistical inference. The application of methods from statistical physics to complex problems in statistical inference is a burgeoning area of research. Many algorithms from statistical physics, both deterministic and stochastic, are becoming popular in statistics. Here we focus on a class of stochastic algorithms called *Markov Chain Monte Carlo* (MCMC).

A first order Markov chain is a series of random variables, X^0, X^1,..., X^n, where the probability distribution from X^n depends on X^0, X^1,..., X^{n-1} only through the value of X^{n-1}. That is, X^n is independent of the past history given X^{n-1}. Markov chains can be described in terms of transition probabilities and states. In other words, the probability of a Markov chain being in state x at time n+1 is given by

$$P_{n+1}(x) = \sum_{\tilde{x}} P_n(\tilde{x}) T_n(\tilde{x}, x) \qquad (3.15)$$

where $T_n(\tilde{x}, x)$ is the transition matrix that defines the probability of moving from state $\tilde{x}$ to state x. The distribution in (3.15) is said to be stationary if it persists forever once reached, that is, $P_{n+1}(x) = P_n(x)$. If $\pi(x)$ is a stationary distribution, then (3.15) can be rewritten as

$$\pi(x) = \sum_{\tilde{x}} \pi(\tilde{x}) T_n(\tilde{x}, x).$$ (3.16)

If a Markov chain satisfies certain regularity conditions [Feller 1968], then it converges to a unique stationary distribution. We can construct a Markov chain with a specified Boltzman distribution as its stationary distribution by ensuring that the transition probabilities satisfy a condition known as *detailed balance* [Neal 1993], also known as local reversibility. Detailed balance ensures that at equilibrium, transitions from any state x to any other state x' are balanced probabilistically by transitions from x' back to x. More formally, a Markov chain satisfies detailed balance if

$$\pi(x) T(x, x') = \pi(x') T(x', x)$$ (3.17)

It is straightforward to verify that a chain satisfying (3.17) has stationary distribution $\pi(x)$. Additional conditions are required on the transition probabilities to ensure that the chain converges to the distribution $\pi(x)$ from any initial condition [Feller 1968]. Detailed balance is a stronger condition than necessary for convergence to a stationary distribution. That is, a Markov chain may converge to a stationary distribution without detailed balance holding. However, detailed balance gives a simple recipe for designing algorithms that converge to a stationary distribution specified up to a normalization constant (i.e., a Boltzman distribution).

The two most common MCMC algorithms are the Gibbs and Metropolis-Hastings algorithms. The simplest of the two is the Gibbs sampler. The Gibbs sampling algorithm works by sampling from and replacing values from a variable conditional upon the current values of all the other variables. The algorithm begins by selecting a variable to

sample from. All remaining variables are set to arbitrary values. A sample is taken from the selected value conditional upon the values of the remaining variables. The variable of interest is then set to the sampled value and a new variable is selected for sampling. (In some versions of the Gibbs sampler, variables are chosen for sampling in a predetermined order: in other versions selection for sampling is random.) The process is repeated until some stopping criterion is met. Gibbs sampling has been successfully implemented in pattern recognition applications [Geman and Geman 1984] and inference in Bayesian networks [Pearl 1988].

The MCMC algorithm used exclusively herein is the Metropolis-Hastings algorithm [Metropolis, Rosenbluth et al. 1953], [Hastings 1970]. The Metropolis-Hastings algorithm samples from a joint distribution by repeatedly generating random changes to the variables in the joint distribution and then accepting or rejecting the changes in a way that preserves detailed balance. In this case the transition probabilities of the Markov chains consist of a proposal distribution and acceptance probability

$$T(x,x') = S(x,x')A(x,x') \tag{3.18}$$

where $S(x,x')$ is the proposal distribution and $A(x,x')$ is the acceptance distribution.

The proposal distribution is used for generating the next candidate state, x^*. It can be as simple as adding a sample from a normal distribution to the current real-valued state or as complex as randomly adding or deleting arcs in a Bayesian network structure. After generating the candidate state, x^*, the state is evaluated and accepted probabilistically. The acceptance distribution is given by

$$x' = \begin{cases} x^*, u < A(x, x^*) \\ x, otherwise \end{cases} \qquad (3.19)$$

where u is a draw from a uniform random distribution and the acceptance probability is given by

$$A(x, x') = \min\left(1, \exp\left(-\frac{(E(x') - E(x))}{T}\right)\frac{S(x', x)}{S(x, x')}\right) \qquad (3.20$$

To show detailed balance holds, let $P(x) = \exp\left(\frac{-E(x)}{T}\right)$. Then from (3.18):

$$P(x)T(x, x') = P(x)S(x, x')A(x, x')$$

$$P(x)T(x, x') = \exp\left(\frac{-E(x)}{T}\right)S(x, x')\min\left(1, \exp\left(\frac{-(E(x') - E(x))}{T}\right)\frac{S(x', x)}{S(x, x')}\right)$$

$$P(x)T(x, x') = \min\left(\exp\left(\frac{-E(x)}{T}\right)S(x, x'), \exp\left(\frac{-E(x')}{T}\right)S(x', x)\right)$$

$$P(x)T(x, x') = P(x')S(x', x)\min\left(\exp\left(\frac{-(E(x) - E(x'))}{T}\right)\frac{S(x, x')}{S(x', x)}, 1\right)$$

$$P(x)T(x, x') = P(x')T(x', x) \qquad\qquad\qquad \blacksquare$$

Thus, the chain satisfies detailed balance as defined in (3.17).

From (3.20) it can be seen that the acceptance probability is higher when the probability of the new state (which is proportional to $\exp(-E(x^*)/T)$) is higher, and when the probability of a transition from x^* to x is higher. That is, new states are more likely to be accepted if they are probable, and if the state being left is "easy" to get back to. Note that Gibbs sampling is a special case of component-wise Metropolis-Hastings sampling,

where the proposal distributions is the probability of the variable being sampled given the other variables, and the acceptance probability as computed by (3.20) is equal to 1.

Markov Chain Monte Carlo algorithms are founded on the first principles of probability theory. They have the advantage that if detailed balance holds, the algorithm guarantees that the Markov chain will converge to the stationary distribution. Once the Markov chain converges, samples can be taken from the desired distribution and estimates can be made of modes, mean, variance, etc... A main drawback of MCMC is that convergence can be slow and difficult to recognize. A chief reason for slow convergence, and for the inability to recognize when convergence has been achieved, is slow mixing. A Markov chain is said to "mix well" when it moves rapidly through the state space, traversing all regions of the state space in a short time. A chain that mixes well will converge rapidly, and it will not take long to obtain samples that can be treated as independent realizations of the distribution of interest. An additional disadvantage from an optimization perspective is that once the algorithm finds a mode, it may take a while to find the local optimum. It's important to note here that MCMC is not designed to find an optimum but to sample from a probability distribution. When the temperature, T in (3.20), is high, the distribution will be more likely to wander over search space than to climb the local hill. For this reason, in optimization problems MCMC algorithms are modified to "anneal" (another concept from statistical physics). The system is run at a high temperature, to find "good" regions of the search space. Then it is slowly "cooled," which makes it behave more and more like a local-hill climbing algorithm. As the temperature nears zero, it climbs to the nearest local optimum and stops. If annealing is

sufficiently slow, there is a high probability that the local optimum at which it stops is near the global optimum [Kirkpatrick, Gelatt et al. 1983].

The Metropolis-Hastings algorithm has been used extensively in statistical inference applications [Gilks, Richardson et al. 1996]. It has also successfully made its way into applications in artificial intelligence research [Geman and Geman 1984], [Madigan and York 1995], [Neal 1993], and [Pearl 1988].

3.3.2 A Markov Chain Monte Carlo Algorithm for Learning Bayesian Networks from Incomplete Data

One approach to learning Bayesian Networks from Incomplete data is to set up two Markov chains for sampling from the incomplete data and Bayesian network structures. This approach is very similar to the evolutionary algorithm approach discussed in Section 3.2. With this approach, there will be two proposal distributions, S_1 for incomplete data and S_2 for structures. The energy term, $E(x)$, in (3.20) is replaced by the log Bayesian Dirichlet score, (2.21).

The missing data and Bayesian network structures have the same representation as given in Section 3.2 for evolutionary algorithms. The proposal distributions are both equivalent to the mutation operators from the evolutionary algorithms. Changes are made to the missing data by choosing cells to modify from the string of missing data and then randomly assigning a value from the set of values (less the current value of the cell) for the variable the missing value was originally assigned in the dataset. A cell is selected for modification with probability p_m. A value is assigned to the selected cell with probability p_v, where p_v is derived as the uniform discrete distribution of selecting a

value from the set of values of the variable associated with the selected cell, less the current value of the cell. For example, suppose the cell is associated with variable V_i which consists of the set of values {1, 2, 3, 4, 5). Suppose the current value in the cell is 4. The cell will be modified by a random selection from the set {1, 2, 3, 5}. The proposal distribution for moving from state x to state x' is then

$$S_1(x, x') = p_m^N p_{V_i},$$
(3.21)

where N is the number of missing data cells and V_i is the set of values in V_i less the current value in the cell. The proposal distribution for the structure consists of selecting a variable with equal probability on each arc and then either adding an arc, deleting an arc, or reversing an arc all with equal probability. To add an arc, the algorithm must select a variable that is not a current parent of the variable under consideration. If there are N variables in the dataset and the current variable has M parents, then the probability of adding a new arc (parent) is $\dfrac{1}{N - M - 1}$. To delete an arc, a variable from the parents of the current variable must be selected for deletion. Assign each parent equal probability of being deleted, then the probability of deleting an arc is $\dfrac{1}{M}$. Finally, the process of reversing an arc is the same as deleting an arch from the current variable and then adding the current variable as a parent of the deleted arc. Therefore, the probability of reversing an arc is the same as the probability of deleting an arc since the arc to reverse must already exist in the current variable. Let the probability of adding, deleting, or reversing an arc be drawn from a uniform discrete distribution, $U_d(y=3)$, where if $U_d(y=3)=1$ then add an arc, $U_d(y=3)=2$ then delete an arc, and $U_d(y=3)=3$ then reverse an arc.

$$S_2(x,x') = \begin{cases} \dfrac{1}{N-M-1}, when U_d = 1 \\ \dfrac{1}{M}, when U_d = 2 \\ \dfrac{1}{M}, when U_d = 3 \end{cases} \tag{3.22}$$

Substituting (3.21) and (3.22) for S(x,x') into (3.18) and (3.20) and letting E(x) represent the log Bayesian Dirichlet metric

$$S(x,x') = S_1(x,x')S_2(x,x')$$

$$A(x,x') = \min\left(1, \exp\left(-\frac{(E(x')-E(x))}{T}\right)\frac{S_1(x',x)S_2(x',x)}{S_1(x,x')S_2(x,x')}\right)$$

$$T(x,x') = S_1(x,x')S_2(x,x')A(x,x') \qquad\qquad \blacksquare$$

Detailed balance holds as shown in Section 3.3.1 above.
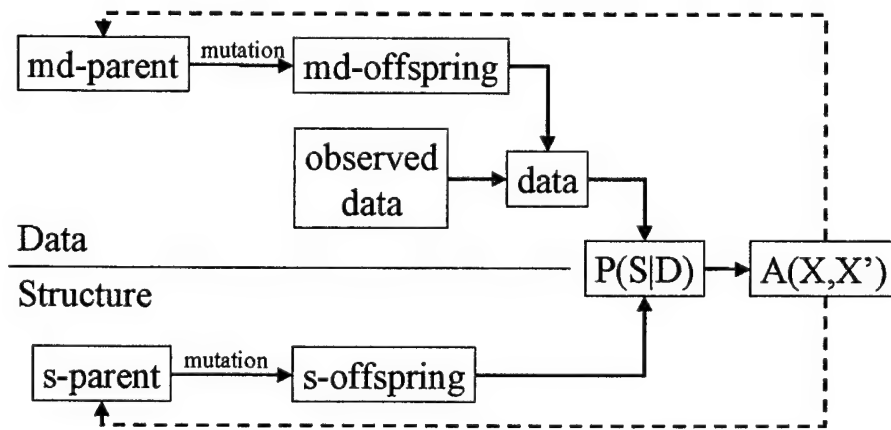
Figure 3-7 MCMC Algorithm for Learning BN from Incomplete Data

The algorithm samples from two proposal distributions simultaneously: the missing data and network structure. Further, the algorithm below samples from multiple

parallel chains. Given enough iterations, the MCMC will converge to the unique stationary distribution of the network structure given the data. The algorithm is given in Figure 3.7.

3.4 Learning Framework Revisited

This chapter gave an overview of the stochastic search methods used in this research. It also completed the instantiation of the learning framework object model used to test the first hypothesis.

- Phenomena: Stochastic observations with missing values.

- Prior Knowledge: Vague; number of variables in the network are known, but the structure is uncertain.

- Goal: Generalization, learn Bayesian networks.

- Constraints: Space and time.

- Assumptions: Missing data are MAR, all structures are equally likely, parameters are discrete.

- Representation:

  - External: Bayesian Networks.

  - Internal: Genotype--string of values for missing data and adjacency list for structure.

- Metric: Bayesian Dirichlet.

- Search:

- Stochastic population-based with global information exchange—EA.

- Stochastic   population-based   with   local   information   exchange-MCMC.

Chapter 4 EVOLUTIONARY MARKOV CHAIN MONTE CARLO

The second hypothesis of this thesis is that the use of global information will speed convergence in MCMC algorithms. A problem noted in the previous chapter is that MCMC algorithms converge slowly to the stationary distribution. Researchers have tried many ad hoc approaches to help speed up convergence. From the learning framework of Chapter 1, a MCMC algorithm with multiple chains is a population-based algorithm with local information exchange. This means that a new state is proposed by making changes to the current state using only information about that state. I propose and test in this research that by taking advantage of global information available in the population, one can speed up convergence to the stationary distribution. I take two approaches to using global information. The first is to use crossover operators from EA to exchange information between two states. I refer to this as the canonical Evolutionary Markov Chain Monte Carlo (EMCMC) algorithm. The second approach is to use information concerning the distribution of arc locations in the population to build a proposal distribution for the mutation operator. The proposal distribution comes from the current population only and can be considered a meta-MCMC in that its distribution depends only on the current population. I refer to this approach as adaptive mutation.

This chapter begins with a brief comparison of the EA and MCMC algorithms. I briefly discuss some of the advantages and problems with each algorithm. Next I describe in some detail the canonical EMCMC and explain why it is both an EA and a

MCMC. The final section discusses the adaptive mutation operator which can be used in the standard MCMC, EA, or the canonical EMCMC. When combined with the MCMC it turns a search that relies solely on local information into a search that takes advantage of global information.

4.1 Comparison of Evolutionary Algorithms and Markov Chain Monte Carlo Algorithms

The main goal of evolutionary algorithms is to search over a large solution space for areas that contain potentially good solutions and exploit those areas to find a "good" solution to a particular problem. As discussed in Chapter 3, an EA can be viewed as searching for local maxima and sampling from the ones with the largest basin of attraction. In an inference problem, then we might consider designing an EA to search for and sample from modes of the distribution of interest. Note that the mode with the most mass is not necessarily the mode with the highest peak. In fact, modes with sharp peaks and little variance are difficult for most search algorithms. Recall from Chapter 3, Vose showed that, in the limit, a simple genetic algorithm would evolve to the fixed point with the greatest basin of attraction. Once an evolutionary algorithm finds such a fixed point, the entire population tends to be attracted to the point, leading to a homogenous population. In essence sampling (exploration) loses out to exploitation once the algorithm finds a mode with a large basin of attraction. For complex, multimodal search spaces the evolutionary algorithm may very well converge to a mode prematurely, leaving modes with better solutions unexplored. The evolutionary algorithm community has tried many approaches to alleviate this problem such as niching [DeJong 1975],

speciation [Spears 1994] and adaptive mutation [Kitano 1990]. To date, there is no agreed upon approach, but there are many promising prospects.

Many empirical studies have supported the use of evolutionary algorithms on problem domains with large, multi-dimensional, multi-modal search spaces. However, there is very little theory, beyond that for the canonical algorithms. This means for most practical problems, researchers must rely on heuristics and empirical studies. There is no guarantee that a specific set of parameters will lead to convergence to a local optimum.

The goal of Markov Chain Monte Carlo algorithms is to sample from the entire posterior distribution. Markov Chain Monte Carlo algorithms do not necessarily find the global mode, but they spend most of their time sampling from modes with the most mass. Markov Chain Monte Carlo algorithms are not typically used for optimization problems, but are important techniques for estimating posterior distributions. However, simulated annealing, a modified version of the MCMC algorithm, has become a very popular optimization algorithm. An important contribution MCMC brings to the search problem is that it is based on the first principles of probability theory—it is guaranteed to converge to the stationary distribution of interest. Thus unlike evolutionary algorithms in which the long term behavior is extremely difficult to analyze, an MCMC algorithm can be designed to converge to a Boltzman distribution of interest. If the objective is optimization then annealing can be used to gradually change the focus from exploration to exploitation. Thus, MCMC algorithms have the advantage over EAs that well established theory from statistical physics can be used to analyze their long run behavior.

While sampling from the stationary distribution includes modes and certainly the mode containing the global maximum, in real applications the MCMC may never sample the true global maximum or even local maximum. One reason for this behavior is that most complex search spaces are essentially continuous and the probability of sampling a local maximum point is very close to zero. Also, the stationary distribution is the area where free energy is minimized. Most samples are drawn from fixed points that represent large basins of attraction. These fixed points are the areas where energy (fitness) less entropy is at its minimum.

Because the MCMC is sampling from large basins of attraction, they tend to stay in these areas for a relatively long time. This leads to a problem referred to as poor mixing, meaning the MCMC is continuing to sample from the same areas and not escaping to sample from other parts of the distribution. Poor sampling can lead to slow convergence to the stationary distribution. Research into improving convergence is a very active area in MCMC. Approaches include reparameterization, random and adaptive direction sampling, and modifying the stationary distribution [Gilks and Roberts 1996]. Researchers have also made large improvements using hybrid algorithms such as the one described by Neal [Neal 1993].

4.2   Combining evolutionary and Markov Chain Monte Carlo

Given the ability of evolutionary algorithms to exchange information in order to improve fitness, it seems reasonable to believe taking the same approach with MCMC will improve the overall fitness of MCMC chains. This may lead to better mixing, faster convergence to the stationary distribution, and possibly to sampling from individuals with

higher fitness overall. For evolutionary algorithms a MCMC approach may also lead to a more diverse population even after convergence. This is because the MCMC is sampling from the stationary distribution and doesn't converge to a few modes or a single fixed point. In addition, an MCMC based evolutionary algorithm is based on the first principles of probability, providing the basis for making theoretically sound statements about the landscape and solutions discovered in the landscape. Adding an annealing schedule may lead to an algorithm that discovers local optimal solutions.

Algorithms that satisfies the conditions of both an evolutionary algorithm as described by equation (3.1) and the conditions for an MCMC as described in Section 3.3 are referred to as Evolutionary Markov Chain Monte Carlo (EMCMC) algorithms. A previous approach for enhancing MCMC with evolutionary algorithm concepts is given below.

4.2.1 Holmes and Mallick

Holmes and Mallick [Holmes and Mallick 1998] combined genetic algorithm approaches with Markov Chain Monte Carlo to increase mixing. Their approach was explicitly tailored for real valued functions. They took the following approach. Randomly generate an initial population of parallel chains, C. At time t, t=1,2,...,n, select a single member, C_t^a, of the population to update. Draw a random number from u = U(0,1). If u<R, where R is a preset parameter, then propose a new state

$$C_{t+1}^{a^{\bullet}} = C_t^a + N_p(0, \sigma_m^2)$$

where p is the dimension of the state space. The proposed state is accepted with probability

$$A(a^*, a) = \min\left\{1, \frac{\pi\left(C_{t+1}^{a^*}\right)}{\pi\left(C_t^a\right)}\right\}$$

where $\pi(x)$ is the probability of state x.

When u>R, they propose a crossover operator where two parents are drawn from the current population, C_t^i and C_t^j where a ≠ i ≠ j. These individuals are selected based on their fitness, possibly using fitness proportional, rank, or tournament selection. The authors used fitness proportional selection. The two parents mate, via uniform crossover to produce one offspring, C_t^x.

The next step is to propose a new state using two types of moves. The first is a move along the direction $\left(C_t^x - C_t^a\right)$ and the other is a reflection of C_t^a about C_t^x. The decision of which move to take is based on a random draw u=U(0,1). If u<F, where F is a preset parameter, then

$$C_{t+1}^{a^*} = C_t^a + 2\left(C_t^x - C_t^a\right)$$

and if u>F then

$$C_{t+1}^{a^*} = C_t^a + \alpha\left[\left(C_t^x - C_t^a\right) / \left\|C_t^x - C_t^a\right\|\right]$$

where $\alpha \sim N(0, \lambda_t)$ and $\|\cdot\|$ is the Euclidean norm. The variance term, λ_t is an adaptive term based on the Euclidean norm between the parents of the offspring. The probability of accepting the new state is as defined above for the mutation.

Holmes and Mallick demonstrated this algorithm on two very difficult high dimensional problems: parameter estimation for a neural network regression model and knot selection for a spline interpolant. They showed that the evolutionary based approach converged quickly to and consistently sampled from higher density areas. In addition, they showed that the evolutionary approach proposed much larger changes than the canonical Metropolis-Hastings sampler did.

These results are encouraging and lead one to believe that by introducing concepts from evolutionary algorithms into MCMC, important improvements can be made in chain mixing and sampling from higher areas of mass.

4.2.2 The Algorithm

The approach described below is just one of many that can be applied. This algorithm may be thought of as the canonical EMCMC. The algorithm is given in Figure 4.1. The first step is to generate an initial population. The population is assigned fitness based on some measure of fitness appropriate for the problem domain. The algorithm then iterates through the following steps until some stopping criterion is reached. The stopping criterion can be a maximum number of generations, a measure of convergence, or any other appropriate metric. Next EMCMC will modify each individual as follows. Select two individuals at random (one can also use other criteria such as ranking or probabilistic selection based on fitness) two individuals. Draw from a uniform distribution, $U(0,1)$ and if $U>P_c$, where P_c is the probability of crossover, then perform crossover. Otherwise perform mutation on each individual. If the parent individuals are

selected for crossover, then they are considered jointly for acceptance. Accept both children with probability,

$$A(X_1X_2, X_1'X_2') = \min\left(1, \exp\left(\frac{\left(E(X_1')+E(X_2')\right)-\left(E(X_1)+E(X_2)\right)}{T}\right)\frac{S\left(X_1'X_2', X_1X_2\right)}{S\left(X_1X_2, X_1'X_2'\right)}\right) \quad (4.1)$$

where $S\left(X_1'X_2', X_1X_2\right)$ is the proposal distribution from the state containing the two parents to the state containing the two offspring and $S\left(X_1X_2, X_1'X_2'\right)$ is the proposal probability for moving from the state containing the offspring back to the one containing the original parents. If crossover is not selected then the two parents are treated separately as in standard MCMC
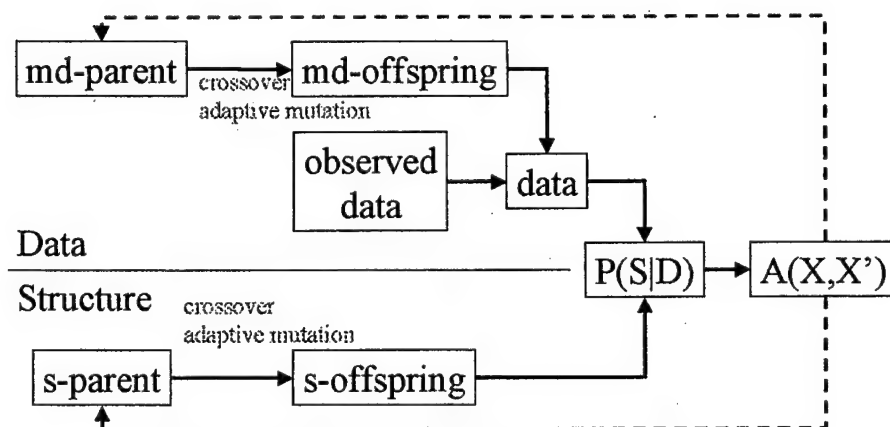
Figure 4-1 EMCMC Algorithm for Learning BN from Incomplete Data

4.2.3 Why EMCMC is an Evolutionary Algorithm

Recall from equation (3.1) that Back defines an evolutionary algorithm as an eight-tuple with the parameters defined therein. The EMCMC can be shown to be an evolutionary algorithm by describing how each of these eight parameters is defined for an EMCMC. An evolutionary algorithm consists of a population of individuals, I. By definition, an EMCMC consists of a population of individuals, I. The fitness function, Φ is the $E(X)$ term in the acceptance distribution. The selection parameter, s, is a random draw without replacement for selecting parents (pre-selection) and the acceptance distribution for determining which individuals live to the next generation (post-selection). The canonical EMCMC has the same set of genetic operators, Ω, as a canonical evolutionary algorithm. Of course, there are many possibilities for additional genetic operators such as the adaptive mutation operator defined below. The EMCMC consists of a set of parents of cardinality μ, and a child population of cardinality λ. For the canonical EMCMC $\mu = \lambda$. It may be possible to make significant performance improvements by defining adaptive birth and death operators that allow the population size to vary from generation to generation in a way that satisfies detailed balance on a modified joint distribution which includes the sample size. This is a subject for future research. The terminator function, ι, can be as simple as a maximum number of iterations, a measure of convergence, or possibly a combination of both. Finally, the process, Ψ, of transforming the current population to the next population is a function of the above parameters.

4.2.4 Why EMCMC is a Markov Chain Monte Carlo Algorithm

To show the EMCMC is an MCMC it is sufficient to show that the algorithm meets the detailed balance criterion as defined in equation (3.16). This means that after making and accepting a change to an individual, we can get back to the original parent or set of parents in a single step. More formally the probability of proposing and accepting a new state X' from state X is equal to the probability of proposing and accepting the original state X from the new state X'. The EMCMC meets this criterion as follows.

When two parents are selected for crossover, they exchange information. They are both accepted or rejected based on their joint fitness. Take the Bayesian network structure as an example. Figure 3.5 shows two individuals and the selected crossover points along with the two proposed (child) networks. From the figure it is trivial to show that if in the next iteration the child networks are selected for crossover and the same crossover points are selected, then the two original networks are generated. Since the probability of selecting any two individuals as parents and selecting crossover points are both uniform, the proposal distributions can be ignored, their ratio is 1, and the acceptance probability simplifies to

$$A(X_1 X_2, X_1' X_2') = \min\left(1, \exp\left(\frac{\left(E(X_1') + E(X_2')\right) - \left(E(X_1) + E(X_2)\right)}{T}\right)\right) \quad (4.2)$$

for crossover since $S\left(X_1' X_2', X_1 X_2\right) = S\left(X_1 X_2, X_1' X_2'\right)$.

When the two parents are not selected for crossover, the algorithm treats each parent separately. Detailed balance was shown to hold for mutation in Section 3.3.

4.3 Adaptive Mutation

The MCMC and canonical EMCMC mutate individual genes by random sampling. I'll refer to the substructures or parameters modified by the mutation operator as genes to be consistent with evolutionary algorithm terminology. Since the MCMC population converges to a stationary distribution, it appears feasible that the current population contains information that can be used for mutation decisions for producing the next generation. For example, take the Bayesian network structure example. An individual in the population is represented by a structure consisting of nodes and arcs. The only difference between the individuals is the placement and direction of the arcs. As the population converges to a stationary distribution, it seems reasonable that the distribution will have more arcs between highly dependent nodes and fewer between conditionally independent nodes. Thus to improve acceptance probabilities and thus speed convergence, the proposal distribution for mutation can be a function of the distribution of arc placements in the structures. Likewise, one should use the distribution over the population of missing values to improve convergence as the distribution will approach the stationary distribution of missing values.

The approach taken herein is to define the distribution of arc placements and missing values as a function of the current population. Previous populations do not directly influence the structure and missing data proposal distributions. Let α be the parameter representing the distribution of arc placements and M the arc placement in the current population. The distribution of arc placements is defined as

$$p(\alpha|M) = \frac{p(M|\alpha)p(\alpha)}{p(M)} \qquad (4.3)$$

where the likelihood $p(M|\alpha)$ is multinomial whose parameters are counts of arcs between pairs of variables, the prior $p(\alpha)$ is the natural conjugate prior of the multinomial, Dirichlet distribution whose hyperparameters are prior counts of arcs between pairs of variables, and the posterior $p(\alpha|M)$ is a Dirichlet distribution. The distribution can be thought of as a two by two matrix where the rows are children and the columns are parents. The value in cell d_{ij} is the probability of an arc from the variable represented by column j to the variable represented by row i, where $d_{ii}=0$.

The next state is proposed as follows. A random draw is taken from (4.3) that corresponds to a cell in the arc distribution matrix. The cell entails a parent-child pair, where the child is the gene. If the parent is in the allele of the gene then the arc is deleted with probability $(1-p_a)$, where p_a is the probability in the cell. If the parent is not in the allele of the gene then an arc is added with probability p_a.

Likewise, the distribution of missing values can be defined as follows. Let β be the parameter representing the distribution of missing values and N the distribution of missing values in the current population. The posterior distribution of missing values is defines as

$$p(\beta|N) = \frac{p(N|\beta)p(\beta)}{p(N)} \qquad (4.4)$$

Since (4.3) and (4.4) only depend on the current population, it is a Markov Chain. In fact, it can be thought of as a meta-MCMC. As the chains converge to the stationary distribution, (4.3) and (4.4) converge to the respective stationary distributions. As (4.3)

converges each mutation adds arcs between highly correlated variables and deletes arc from conditionally independent variables. As (4.4) converges the missing values represented in the population are samples from the distribution of the phenomenon being modeled.

Chapter 5 EMPIRICAL APPROACH

In chapters 3 and 4, three stochastic search algorithms were described. Implementations of these algorithms were designed for the problem of learning Bayesian networks from datasets with incomplete information. The algorithms search over structures and completions of the missing data, and the resulting networks are scored using the Bayesian Dirichlet scoring metric. The evolutionary algorithm uses the Bayesian Dirichlet score as the fitness function, and thus evolves toward structures and missing data completions that are probable a posteriori given the observed data. The Markov Chain Monte Carlo and evolutionary Markov Chain Monte Carlo algorithms are both designed to converge in the long run to a stationary distribution given by the posterior distribution of structure and missing data given the observed data. The solutions found using all of these approaches are not necessarily the "true" model that produced the data. In fact, with incomplete information, we can only hope to reduce the uncertainty by selecting a "good" set of models.

This chapter describes the empirical approach used to evaluate this set of algorithms. The overall approach, first to find a good set of parameters for each algorithm and then compare each algorithm's performance, is discussed in Section 5.1. Section 5.2 describes the metrics used for both selecting parameters and comparing the algorithms. The design of the experiments is described in Section 5.4. Finally, the

results from the parameter selection experiments are provided in Section 5.5. Chapter 6 provides the results from the comparative experiments.

5.1 Approach

The empirical study is composed of two parts. First, a set of experiments is conducted on each algorithm to find a "good" set of parameters. No attempt was made to find an optimal parameter set. This would have required finding an optimal design and many more runs than practical. The objective was to find a set of parameters that works well for learning Bayesian networks from incomplete data, and will suffice as parameter settings for conducting the remainder of the experiments. The second set of experiments is conducted to compare the algorithms with each other. One goal of the comparisons is to learn whether any one class of algorithms is an overall better performer than the other two. Algorithms are compared using both the single best model and of the final population of models found by the algorithm in its search. Examining the final population of models is a way to address the question of whether averaging predictions from a population of models outperforms predictions from a single top-scoring model, i.e. the third hypothesis. A final goal is to identify areas, for each algorithm, for potential future improvement and research.

The set of parameters for each algorithm was found by an experiment in which parameters were varied using a factorial design. The experimental design is described in Section 5.3 below. All data for the first set of experiments were generated from a known network. This network has seven variables and each variable is binary-valued. The network will be referred to from here on as the 1x3x3 network and is shown in Figure

5.1. This network was selected because it is particularly applicable for learning hidden variables. Inserting a hidden variable between the second and third layer of the network produces the much-simplified network shown in Figure 5.2.
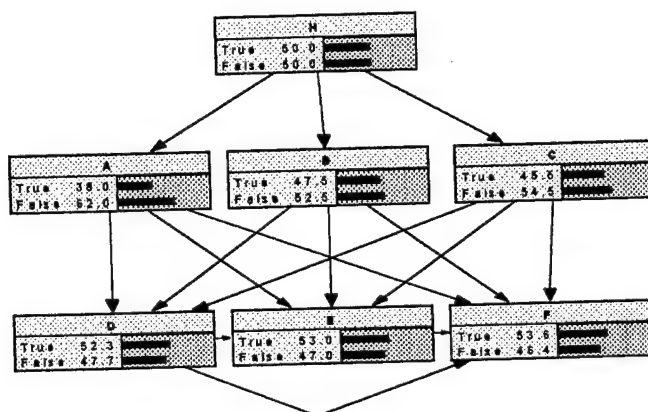
Figure 5-1 1X3X3 Network
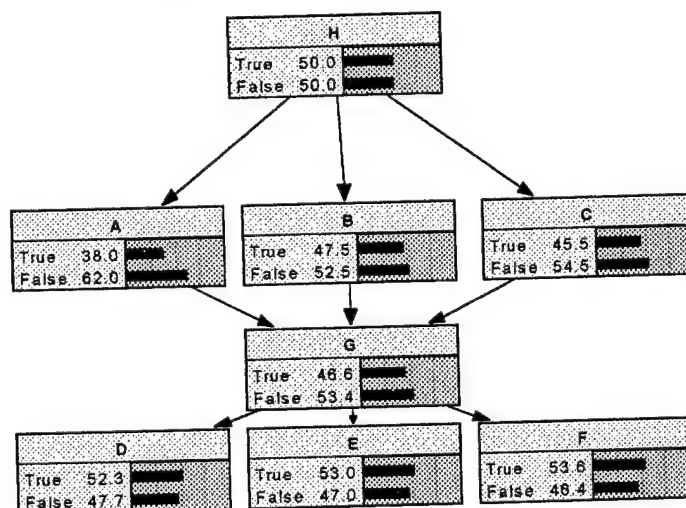
Figure 5-2 1X3X3 Network with Hidden Variable

There are two main reasons to evaluate algorithms using a known network. First, adequate sets of training and test data are easily generated from the network. Second,

results on learned networks (model scores and predictive performance on test data) can be compared with results from the network known to have generated the data.

From the 1x3x3 network, a set of training and test data containing 1000 cases each was generated. A percentage of the training data were then randomly deleted to simulate incomplete data. When finding hidden variables, the entire data column representing the hidden variable was classified as missing. Each experiment of the factorial design consisted of a run of the algorithm to learn networks from the training data. To find the best set of parameters, we used the best network found by the algorithm during the run. We used two performance metrics: the Bayesian Dirichlet score (the logarithm of the joint probability of the network, and the missing data completion, and the log data) and the log loss. In addition, for the Markov Chain Monte Carlo algorithms we measured the number of iterations required until convergence to the stationary distribution. The reason for choosing these metrics is discussed in Section 5.2.

The second set of experiments to compare each algorithm proceeds in a similar fashion. We use the same network as in parameter selection to generate the training and test data. The missing values randomly generated from the training data. The algorithms were run with the parameters found in the first set of experiments. The metrics used to compare the algorithms are the Bayesian Dirichlet, accuracy (classification error rate), log loss, best so far curves, and convergence curves. The algorithms were compared in terms of the best so far network and multiple models. The comparisons using multiple models will use the log loss exclusively. I also ran experiments on a Real-world dataset used to investigate factors influencing whether high school students will attend college

[Sewell and Shah 1968]. Another set of comparative experiments were run using a larger known network used to model administering anesthesia in emergency runs [Beinlich, Suermondt et al. 1989].

5.2 Metrics

5.2.1 Bayesian Score

One natural choice of a metric is the Bayesian Dirichlet score. The Bayesian Dirichlet measures the log posterior probability that the model under consideration fits the prior knowledge and data. It is the logarithm of the joint probability of observed data, missing data, and structure, and therefore is equal to a constant plus the logarithm of the posterior probability of structure and missing data given the observed data. The Bayesian Dirichlet score is used as the fitness function for all of the algorithms explored herein. It is the most common metric used for selecting a single structure from those considered in the search space.

As the number of variables grows the possible number of structures increase exponentially. The large set of possible structures means that even the most likely structures have very low posterior probabilities. For computational convenience, the log of the posterior probability is typically computed. This approach is also more computationally efficient since the log is a series of additions as opposed to products.

Another interesting feature of the log of the Bayesian Dirichlet is the following interpretation. The relative posterior probability is defined as

$$p(D, B_s) = P(D|B_s)P(B_s) .$$

(5.1)

the log probability then is

$$\log p(D, B_s) = \log p(B_s) + \log p(D|B_s) \qquad (5.2)$$

the first component on the right of (5.2) is the log prior probability of the structure and the second component is known as the log marginal likelihood. From the chain rule (2.3) the log marginal likelihood is expressed as,

$$\log p(D|B_s) = \sum_{l=1}^{N} \log p(x_l|x_1,\ldots,x_{l-1},B_s) \qquad (5.3)$$

where the $p(x_l|x_1,\ldots,x_{l-1},B_s)$ is the prediction probability for x_l given model B$_S$ after averaging over the parameters of B$_S$. Dawid makes a strong case, in his prequential framework [Dawid 1984] for using predictive probabilities for model selection. The prequential framework states that it is better to use observed outcomes in building a predictive system than unknown parameters. The prequential framework is a means of developing forecasting systems. The name is derived from combining *probability* and *sequential* prediction. A prequential forecasting system is a rule that assigns a probability P$_{n+1}$ to a set of outcomes $\mathbf{X}^{(n)}$ for the purpose of predicting the future outcome $\mathbf{X}^{(n+1)}$. Any statistical system that meets this very broad requirement is a prequential forecasting system. It's trivial to see that (5.3) is a prequential forecasting system.

In assessing model adequacy, one can use the prequential principle. Given a prequential forecasting system, P for $\mathbf{X}$, the realized outcomes $\mathbf{x}=(x_n)$, and the set of predictions $\mathbf{P}=(P_n)$ which were produced by P, the overall adequacy of the predictive system depends only on the previous prediction $\mathbf{P}$. Using this principle and equation

(5.3), the model with the highest log marginal likelihood is the overall best predictive model.

Further, Dawid describes a *calibration* criterion as a means of comparing prequential probabilities. It states that any sufficient subset of **P** averaged together should agree asymptotically with the average of the observed **x** values [Dawid 1982]. Dawid further shows that if two prequential systems P and Q satisfy the calibration criterion, then $p_n - q_n \to 0$ as $n \to \infty$. In other words, two prequential systems satisfying the calibration criterion make the same forecasts in the limit.

The result from the calibration criterion has some far-reaching implications. (In words, if we can not reject a model based on empirical results then we should keep it.) Since, for any scientific problem, there are potentially many models that can not be rejected based on empirical results, there is no need to search for the one "true" model. One should select a representative model or preferably select multiple models.

Another interesting result from Dawid's exposition of the prequential framework is its relationship with the cross validation (CV) approach used in model selection. Consider the cross validation approach known as *leave-one-out*. The approach is to set aside a single case, x_i, and train the model on the remaining data. The remaining data is known as the validation set, $V_i = \{x_1, \ldots, x_{i-1}, x_{i+1}, \ldots, x_N\}$. The model's predictive performance is then measured using case x_i. Next, set aside another case and repeat the process until all cases have been used in the prediction. The cross validation metric is

$$CV(B_s, D) = \sum_{i=1}^{N} \log p(x_i | V_i, B_s) \tag{5.4}$$

Notice the similarity with equation (5.3). The cross validation approach is a very commonly used practice in model selection. The reason for using a cross validation approach is to avoid over-fitting the data. In fact, the cross validation process still leads to some over-estimation. The reason for this is at each iteration, the training and test cases are interchanged. For instance to estimate $p(x_1|V_1,B_S)$, x_2 is in V_1 and to estimate $p(x_2|V_2,B_S)$, x_1 is in V_2. The prequential approach does not interchange test cases with training cases and thus does not lead to over-estimation [Dawid 1984] [Heckerman 1996].

The primary use, in this research, of the Bayesian Dirichlet is as the fitness function for each of the algorithms. It is the primary metric used to select a single "best" model from the population of solutions from a run of the algorithms. It is also a convenient score to use for the "best so far" curves discussed later. Its variance during runs is an independent variable in the convergence curves, discussed later, and is used to assess diversity in the population of solutions.

5.2.2 Accuracy

Accuracy is a measure of the percentage of cases classified correctly or incorrectly on test data. The percentage of cases incorrectly identified in a test set is referred to as the error rate. There are several approaches for finding the accuracy of a model. If enough data are available, the data can be split into a randomly drawn training set with the remaining data composing the test set. If the data set is small, an acceptable approach is cross-validation. But as noted in Section 5.2.3, cross validation can lead to some over-

fitting the data. For small datasets we may have no other choice since the prequential approach requires large training sets to work effectively.

Accuracy is arguably the most common metric used in machine learning research. Thumbing through the pages of the latest AI conferences is a good indicator of the popularity of this metric. An informal sample of the AAAI98, UAI98, and GP98 conferences, indicated 19 of 35 articles involving some form of machine learning and/or classification used accuracy as a metric [AAAI98 1998], [UAI98 1998], [GP98 1998]. In Michalski's Machine Learning book, six of nine chapters involving performance measures of classification systems used accuracy as a metric. The three that did not use accuracy were approaches focused on reducing model complexity [Michalski and Tecuci 1994]. In the latest book describing the results from StatLog (a research program comparing machine learning techniques in Europe), Michie et al., used accuracy as their primary metric [Michie, Spiegelhalter et al. 1994].

For deterministic classification problems, accuracy is a perfectly acceptable measure. That's because the variable is either classified correctly or not. But for most real-world problems there is uncertainty. Suppose a doctor uses an expert system to infer whether a patient has a given disease from a set of tests. A deterministic system reports a single answer: yes or no. The doctor, and patient, may have some doubt, because tests are known to misclassify at times. In contrast, a probabilistic system gives the doctor information about certainty of its classification. If the system is highly certain, the doctor might trust its answer. On the other hand; if it reports a 0.52 probability that the patient has the disease, the doctor may want to run a few more tests.

The accuracy metric is a noisy metric at best for systems that give probabilistic answers. It cannot distinguish between a correct response on a problem about which the system was highly certain or a correct response on a problem about which the system was unsure. Accuracy measures are computed and reported for comparison purposes, but were not used to determine parameter settings or to make judgments about the relative performance of algorithms.

5.2.3 Log loss

Selecting a model from a set of models is a decision problem. In this context each model can be thought of as a set of hypotheses $\{M_j, j \in J\}$. Given a set of relevant observations, D, one may make a set of precise conditional probability statements, Q over the set of hypotheses M

$$Q = \left\{ \mathbf{q} \equiv (q_j, j \in J); \ q_j \geq 0, \ \sum_{j \in J} q_j = 1 \right\},$$

where each q_j is the probability (belief) conditional on the data for the hypothesis M_j being true. The set (q, M_j) is known as the consequence and $\{(\mathbf{q},M_j)|M_j, j \in J\}$ is the action associated with the choosing $\mathbf{q}$ [Bernardo and Smith 1994].

In order to make a decision, one can generate a rule that assigns a value to how "good" or "bad" is an outcome of a particular decision. This rule is a function of the conditional probabilities and the hypotheses and is known as the *utility*, $U(\mathbf{q},M_i)$ [O'Hagan 1994]. It is particularly convenient in the context of model selection to use *loss functions*, $L(\mathbf{q},M_i)$, instead of a utility. A loss function is defined as the difference

between the utility of the best possible outcome and utility actually received [Press 1989].

Utility functions and loss functions are known as scoring rules or scoring functions.

> **Definition 5.1.** **(Score function) [from Bernardo, [Bernardo and Smith 1994]].** A score function u for probability distributions $q=\{q_j, j \in J\}$ defined over a partition $\{M_j, j \in J\}$ is a mapping which assigns a real number $u\{q, M_j\}$ to each pair (q, M_j). This function is said to be **smooth** if it is continuously differentiable as a function of each q_j.

The score function described above specifies a problem that can be considered in terms of an expected utility. The decision-maker can then choose the **q** that maximizes the expected utility. It is possible to assign a utility function that rewards dishonest reporting of beliefs. For such a utility function, the probability distribution **q** that maximizes the decision maker's utility is not the same as the decision maker's actual probability distribution for the phenomenon. In "pure inference" problems it is desirable to use a utility function that is maximized when the probability distribution **q** reported by the decision maker reflects the decision maker's true probabilities.

> **Definition 5.2. (Proper Score function) [from Bernardo, [Bernardo and Smith 1994].** A score function u is proper if, for each strictly positivie probability distribution $p=\{p_j, j \in J\}$ defined over a partition $\{M_j, j \in J\}$,
>
> $$\sup_{q \in Q} \left\{ \sum_{j \in J} u(\mathbf{q}, M_j) p_j \right\} = \sum_{j \in J} u(\mathbf{q}, M_j) p_j$$
>
> where the supremum, taken over the class Q of all probability distrubutions over $\{M_j, j \in J\}$, is attained if, and only if, $\mathbf{q} = \mathbf{p}$.

The log loss scoring function is such a proper scoring rule commonly used for probabilistic learning algorithms. The log loss score is used to measure how well a model predicts an observation. For a variable X on case i, the log score is given by

$$L(\mathbf{q}, M_i) = -\log p(x_i) \tag{5.5}$$

where $p(x_i)$ is the probability that the model assigns, given the values of the observed variables other than X on case i, to the actual observed value x_i. It has range $(0,\infty)$, where the smaller values are preferred. When the prediction is certain, the log score is zero and as the prediction approaches P=0, the log score approaches infinity. As an example, suppose for case C_i, the model predicts the correct outcome with probability P=0.95. The log score will be $L(q,M_i)$=0.05. If the probability is P=0.05, the corresponding log score is $L(\mathbf{q},M_i)\cong 3$.

When selecting from a set of models, it is customary to test each model's predictive performance from a set of test data. The metric in this case is the average log score for cases in the test sample. The log score of each case is the log of the joint probability of each case. For large models, even the most probable case can have a low probability prediction. This is because there are a large number of possible cases and the joint probabilities must sum to one, $\sum_{i=1}^{N} p_i = 1$.

The log loss function is intrinsically tied to information theory. Let X be a random variable with density f(X) and Q is the set of all density functions q(X) (for discrete random variables, f(X) and q(X) are mass functions and the integrals below are replaced by sums). If X is the observed value, then the loss from predicting according to the density $\mathbf{q}$ is given by $L(\mathbf{q},X) = -\log q(X)$. The expected log loss is

$$E(L(\mathbf{q}, X)) = -\int_{-\infty}^{\infty} f(X) \log \mathbf{q}(X) dX \qquad (5.6)$$

The expected log loss is minimized by predicting according to the true density f(X) and is defined as

$$H(f) = -\int_{-\infty}^{\infty} f(X) \log f(X) dX \tag{5.7}$$

Equation (5.7) is the entropy of f(X). If one takes the difference of the expected log loss in (5.6) and the minimum expected log loss (entropy) in (5.7) the result is the *Kullback-Leibler* (K-L) *divergence* [O'Hagan 1994]

$$D(f \| \mathbf{q}) = \int_{-\infty}^{\infty} \log \frac{f(X)}{\mathbf{q}(X)} f(X) dX \tag{5.8}$$

The K-L divergence is a measure of the "distance" between two densities. It is not a true distance measure because $D(f \| \mathbf{q})$ is not necessarily equal to $D(\mathbf{q} \| f)$. It can also be thought of in terms of the inefficiency introduced by selecting $\mathbf{q}$ when the true distribution is f.

5.2.4 MCMC convergence

A common problem with Markov Chain Monte Carlo algorithms is they are notoriously slow to converge to the stationary distribution. This can happen when a large basin of attraction is near a starting distribution. The chain will tend to sample in the basin of attraction for a long period of time, only after a long time escaping to sample the remaining distribution. It is very difficult to tell when a single chain is sampling from the stationary distribution. Gelman and Rubin recommend using multiple chains. With multiple chains one can tell visually whether they have jointly converged to a stationary distribution when one can not distinguish the chains. Another method for monitoring

convergence proposed by Gelman and Rubin is based on monitoring the within chain and between chain variance [Gelman and Rubin 1992].

Let ψ represent the summary we are interested in monitoring for convergence. In the case of learning Bayesian networks, ψ is the Bayesian Dirichlet score. In addition let m represent the number of chains and n the length of the chain. The idea is to measure the within chain variance and the between chain variance. If a chain has a small within chain variance but it is far from other chains then the chains have not converged. It is possible that some of the chains have converged but the only way to know for sure is to wait until they all have converged. When all chains have converged, then the between and within chain variances will be very close.

Let B represent the between chain variance and W the within chain variance. Further,

$$B = \frac{n}{m-1} \sum_{i=1}^{m} \left(\overline{\psi}_{i.} - \overline{\psi}_{..} \right)^2 \tag{5.9}$$

where $\overline{\psi}_{i.} = \frac{1}{n} \sum_{j=1}^{n} \psi_{ij}$, and $\overline{\psi}_{..} = \frac{1}{m} \sum_{i=1}^{m} \overline{\psi}_{i.}$

and

$$W = \frac{1}{m} \sum_{i=1}^{m} s_i^2 \tag{5.10}$$

where $s_i^2 = \frac{1}{n-1} \sum_{j=1}^{n} \left(\psi_{ij} - \overline{\psi}_{i.} \right)^2$.

From the variance components B and W, Gelman and Rubin define upper and lower bounds for the actual variance of the chains, var(ψ). The upper bound is an estimate of the unbiased estimate when the chains reach the stationary distribution.

$$\hat{v}(\psi) = \frac{n-1}{n}W + \frac{1}{n}B \qquad (5.11)$$

The lower bound is the within variance component (5.10). As the two estimates converge from the upper and lower bounds, their ratio approaches one. Gelman and Rubin define the measure of convergence for multiple chains in terms of what they refer to as the 'estimated potential scale reduction',

$$\sqrt{\hat{R}} = \sqrt{\frac{\hat{v}(\psi)}{W}} \qquad (5.12)$$

As the chains converge, the scale reduction metric converges from above to 1. From experience, Gelman and Rubin recommend setting a limit for convergence at around 1.1 or 1.2.

5.2.5 Best so far curves

A common metric used to compare evolutionary algorithms is the so-called "best so far" curve. Originally proposed by DeJong, it is a plot of the fitness values of the overall best solution found during a run. For stochastic algorithms such as the ones considered herein, the "best so far" curves are averaged over multiple runs. The best so far curve can tell us several things about an algorithm. First, it measures how fast, on average, the algorithm converges to the highest scoring (best fit) solution. The plot can also give a

feel for algorithms that produce overall better single solutions, by comparing the average fitness values of the "best so far" solutions [DeJong 1975].

5.2.6 Multiple Models metrics

The final metric used in this research concerns the predictive accuracy of multiple models, the third hypothesis presented in Chapter 1. Each of the algorithms discussed produces populations of solutions. It would be fruitful to explore:

1) whether the single "best" model outperforms a set of models, both

    a) within an algorithm and

    b) between algorithms, and

2) whether any of the algorithms produces a set of models that together predict better than those produced by any other algorithm.

Most work in the field of model selection has concentrated on the selection of a single "best" model from a set of models. As described in Sections 5.2.1 and 5.2.3, this is essentially a decision problem of selecting the model with the highest likelihood or minimum log loss. However, one group of researchers, Madigan, Raftery, et al., have studied the performance of multiple models versus a single best model. They suggest and provide some empirical evidence that a "good" set of multiple models may outperform the single "best" model. The reason for this is that the single "best" model is selected as if it were the "true" model, where in fact it is most likely not the true model. There is uncertainty in any model selection procedure. They propose averaging over multiple models to account for the uncertainty in the models [Madigan and Raftery 1994],

[Madigan, Raftery et al. 1994], [Madigan and York 1995], and [Madigan, Raftery et al. 1996].

The typical Bayesian approach to averaging over the models is as follows. Suppose C is the item we are interested in predicting. The predictive probability over all the models is

$$p(C|D) = \sum_{i=1}^{K} p(C|M_i, D) p(M_i|D) \qquad (5.13)$$

where the posterior probability for the model is obtained from Bayes' Rule

$$p(M_i|D) = \frac{p(D|M_i) p(M_i)}{\sum_{j=1}^{K} p(D|M_j) p(M_j)} \qquad (5.14)$$

and the likelihood in (5.14) is

$$p(D|M_i) = \int p(D|\theta, M_i) p(\theta|M_i) d\theta \qquad (5.15)$$

where θ is a vector of parameters for the model.

Unfortunately direct implementation of this approach presents insurmountable problems for applications of any complexity. First, the integral in (5.15) can be very difficult to compute or even to approximate. Second, the number of models, K, in equation (1) can become very large making computation intractable. Madigan, et al., recommend selecting a representative set of models and averaging over the smaller set.

For measuring predictive performance they recommend using the log score described in Section 5.2.3. For multiple models (5.5) is generalized to

$$L(\mathbf{q}, \mathbf{M}) = -\sum_{d \in D^T} \log \left\{ \sum_{\mathbf{M} \in \mathbf{A}} p(d|\mathbf{M}, D^S) p(\mathbf{M}|D^S) \right\} \qquad (5.16)$$

where D^T is the test data, D^S is the training data, **M** is the set of models drawn from the set of all acceptable models **A**.

The MCMC and EMCMC algorithms are designed to converge to independent samples from the posterior distribution we are estimating. Therefore, it is appropriate to estimate (5.16) by a simple unweighted average of the models in the final population.

5.3 Experimental Designs

Factorial experiments were used to determine a set of good parameters for each of the algorithms. Because of time constraints, there was no attempt to find an optimal set of parameters. Instead, the experimental results were used to select a set of parameters that produced good results. Engineering judgement was used in deciding which parameters to use as factors in the factorial designs. This judgement was based on previous experience, literature search, and small test runs. All remaining parameters were set at a pre-selected constant value. Each experiment described below was replicated five times. Two response variables were used to assess algorithm performance: the Bayesian Dirichlet metric and log loss of the best network from a run. In addition, convergence rate was used for the MCMC algorithms.

5.3.1 Evolutionary Algorithm

The evolutionary algorithm described in Section 4.3 has nine controllable parameters. They are selection scheme, probability of crossover for the structure, parameter for uniform crossover of the structure, mutation rate of structure, probability of crossover for the missing data, parameter for uniform crossover for the missing data,

mutation rate for the missing data, population size, and maximum number of generations (stopping criterion). The parameters selected as factors in the design are selection scheme, parameter for uniform crossover for both structure and data, and mutation rate for structure.

The remaining parameters were not selected as factors in the design for the following reasons. The population size was set at 20. This was the largest size that could be accommodated given constraints on computation time. A higher population size may have made a difference, but based on a few trial experiments with population size set at 40 there was no noticeable difference in the outcome. The stopping criterion was set at 100 generations. This decision was based on engineering judgment and a set of trial experiments. While the algorithm was still finding better solutions when stopped, they were not significantly better from 100 generation to 500 generations. The probability of crossover was set at $Pc = 0.65$. This choice was based on engineering judgement, literature search, and trial experiments. DeJong [DeJong 1975] and Goldberg [Goldberg 1989] both recommend crossover rates of around 0.65. From a few trial experiments, this crossover rate appeared to perform well. Finally the mutation rate for the missing data chromosome was set at 0.01. This is actually a higher rate than would be expected. Both DeJong [DeJong 1975] and Goldberg [Goldberg 1989] recommend much lower mutation rates. However, the chromosome length for the missing values is very long, on the order of 1,000 to 2,500 genes. In trial experiments, the higher rate appeared to perform better. The results however were not conclusive and this is a subject of further research.

The factorial design consisted of four factors: selection, parameter for uniform crossover of structures, mutation rate of structures, and parameter for uniform crossover rate for missing data. The selection factor has three levels representing the three selection schemes: fitness proportional selection, rank proportional selection, and binary tournament selection. The remaining factors are all two level factors. Each level was selected to represent a low and high range.

5.3.2 Markov Chain Monte Carlo Algorithm

The Markov Chain Monte Carlo algorithm consists of only two controllable parameters; population size, and maximum number of generations (stopping criteria). The population size was set at 20. There is really no need to vary population size for MCMC since they do not interact in any way. They use local information to propose new states. The size of 20 was selected to be consistent with the other algorithms. The stopping criterion was set at 500 to allow enough time for the chains to converge. With both controllable variables pre-set, there were no experiments needed to select the parameter values.

5.3.3 Evolutionary Markov Chain Monte Carlo

The evolutionary Markov Chain Monte Carlo algorithm is similar to the evolutionary algorithm. It has four controllable parameters: probability of crossover, parameter for uniform crossover of the structure, parameter for uniform crossover for the missing data, population size, and maximum number of generations (stopping criteria). Since the parameters for uniform crossover should perform the same for EMCMC as they did for

the EA, they were set at the values found during the EA experiments. The population size was set at 20 and the stopping criterion was set at 500 generations to allow enough time for the chains to converge.

The most important question to answer for this algorithm is at what rate should two parents exchange information. The only variable factor that controlled information exchange was probability of crossover. A single crossover rate suffices for both the population of structures and the population of missing data. The reason is that when two parents are selected both the structure and the missing values associated with the parents must be used to produce two offspring. Recall that either both offspring are accepted or both are rejected after a crossover operation. The levels selected for crossover rate were 0.05, 0.25, 0.50, 0.75, and 0.95.

5.4 Empirical Results for Parameters

The data from the experimental designs described above were all analyzed using both a frequentist and a Bayesian approach. The frequentist approach used was an analysis of variance (ANOVA). The Bayesian approach was to build a generalized linear model with prior distributions for the parameters. The parameters were then inferred conditional upon the model and the data using Gibbs Sampling. Gibbs Sampling was described in Chapter 3. The software used for the ANOVA was S-Plus© and the software used for the Gibbs Sampling was BUGS© (Bayesian inference Using Gibbs Sampling), downloaded from the MRC Biostatistics Unit, Institute of Public Health, Cambridge, URL= http://www.mrc-bsu.cam.ac.uk/bugs/Welcome.html. An appropriate

set of parameters was then selected by examining the results of both the ANOVA and the Bayesian analysis.

5.4.1 Evolutionary Algorithm

An ANOVA was performed on the results from the factorial design. A fully saturated model was used. This means that the main effects and all combinations of interactions were included in the model. The results are provided in Table 5.1. From the ANOVA, none of the interactions were significant. All main effects were significant except the parameter for uniform crossover for the structure. From the estimated coefficients the recommended levels were binary tournament selection, a mutation rate of 0.05, and parameter for uniform crossover for missing data set at 0.5.

Table 5-1 ANOVA Results for EA

	Df	Sum of Sq	Mean Sq	F Value	Pr(F)
select	2	259992.6	129996.3	160.2522	0.0000000
unifparm	1	195.1	195.1	0.2405	0.6249804
pmparm	1	4431.1	4431.1	5.4624	0.0215075
unifdata	1	4102.0	4102.0	5.0567	0.0268167
select:unifparm	2	636.2	318.1	0.3922	0.6766704
select:pmparm	2	2899.9	1450.0	1.7874	0.1729152
unifparm:pmparm	1	914.1	914.1	1.1269	0.2911080
select:unifdata	2	1378.6	689.3	0.8497	0.4307277
unifparm:unifdata	1	1.6	1.6	0.0020	0.9643027
pmparm:unifdata	1	2022.1	2022.1	2.4928	0.1176605
select:unifparm:pmparm	2	1627.9	813.9	1.0034	0.3704568
select:unifparm:unifdata	2	285.7	142.9	0.1761	0.8387989
select:pmparm:unifdata	2	1484.8	742.4	0.9152	0.4039091
unifparm:pmparm:unifdata	1	1237.8	1237.8	1.5259	0.2197492
select:unifparm:pmparm:unifdata	2	2776.9	1388.5	1.7116	0.1860399
Residuals	96	77875.0	811.2		

The generalized linear model used for the Bayesian inference was limited to main effects and two-way interactions due to software difficulties. This should not be a problem because the ANOVA indicated that none of the interactions were significant. All of the parameter priors were set as noninformative prior and specified as $N(0, 1.0E-4)$.

The Gibbs Sampler was run for 10,000 iterations to guarantee convergence to the stationary distribution. The resulting 95% credible intervals for the model parameters are given in Figure 5.3 and Figure 5.4. As can be seen the selection scheme is by far the most significant contributor to the Bayesian Dirichlet score. The mutation was also significant but did not contribute very much to the response variable. These results are consistent with the results of the ANOVA. Further review of the estimated coefficients from the ANOVA indicate that the mutation rate and parameter for uniform crossover of missing data contribute less than one percent to the Bayesian Dirichlet score.

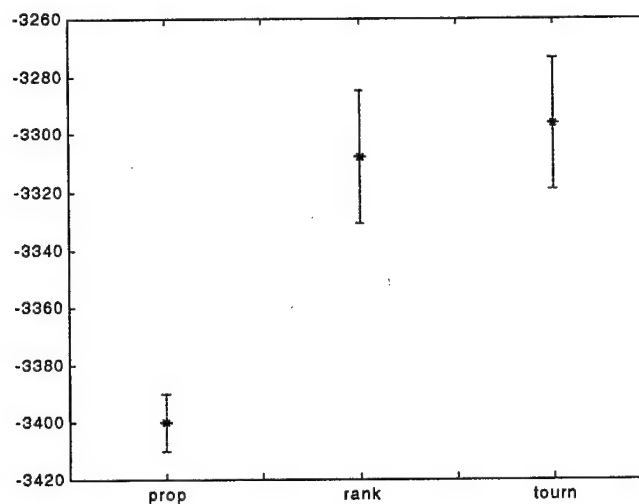

Figure 5-3 95% Credible Interval for Selection Schemes

The real decision then is which selection scheme to use. Proportional fitness is by far the worst performer from both the ANOVA and Bayesian Inference. The best selection scheme from both analyses is the binary tournament selection. The remaining parameters are set as follows: both parameters for uniform crossover were set at 0.5 and

the mutation rate for the structures was set at 0.05. Even though these main effects were
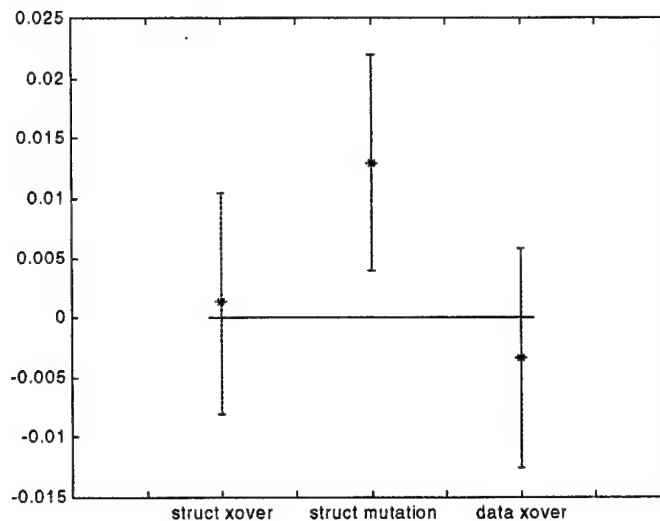not significant, these values produced the best scores from the model.

Figure 5-4 95% Credible Interval for Crossover and Mutation

5.4.2 .Evolutionary Markov Chain Monte Carlo

A Bayesian analysis was conducted on the EMCMC algorithm with crossover rate
as the only factor. Crossover was given five levels; 0.05, 0.25, 0.50, 0.75, and 0.95. The
95% credible interval for rates between 0.05 and 0.75 overlapped for both the Bayesian
Dirichlet and log loss indicating that we can not completely distinguish between these
four levels. The 0.95 crossover rate was significantly worse than the other four rates for
both metrics and was discarded accordingly. Further inspection of the plots in Figure 5.5
show a slight trend from the lower crossover rate to the higher values with the lower rates
being better.

Figure 5.6 shows both the convergence rate (top plot) and the best so far curves
(bottom plot) for each level. The lower three crossover rates (0.05, 0.25, and 0.50) have

similar convergence curves with the 0.05 rate being slightly better. The 0.75 crossover rate did not converge as fast as the three lower rates and the 0.95 crossover rate did not converge at all. The best so far curves were consistent with the results from the 95% credible interval from above.

These results indicate that the lower crossover rate performs better than higher crossover rates. For the comparative studies, the crossover rate was set at the lower level, 0.05.
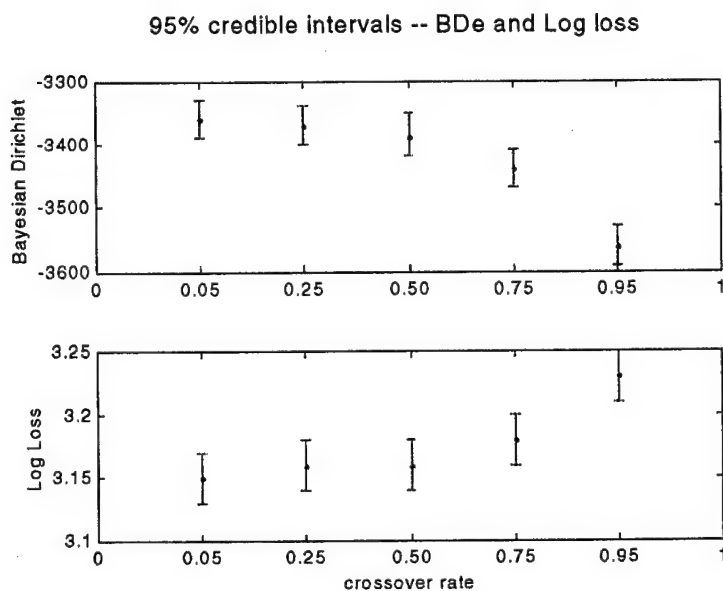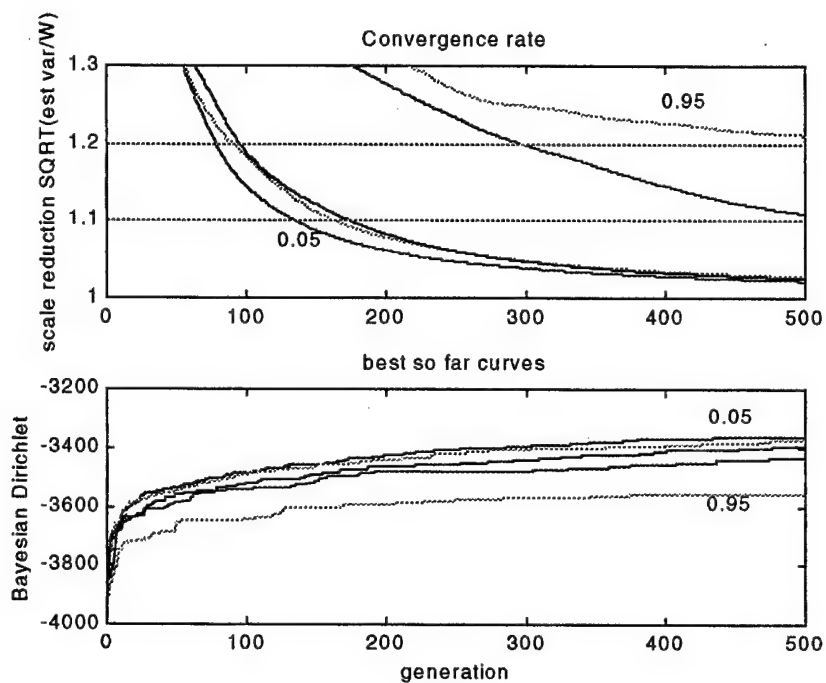

Figure 5-5 95% Credible Intervals for Crossover Rate

Figure 5-6 Convergence Rate and Best So Far for Crossover Rates

Chapter 6 Empirical Results

This chapter provides empirical evidence for the three hypotheses articulated in Chapter 1. I'll use three carefully chosen datasets to demonstrate that stochastic population-based algorithms are able to find Bayesian networks from incomplete data with good predictive power. I'll compare the algorithms and discuss the findings. In addition, I compare them to a simple greedy hill-climbing approach. Then I will show how one can use global information to improve mixing and speed convergence in a MCMC. Finally, I'll provide an example that provides further evidence to existing empirical results that multiple model averaging has better predictive power than using a single "best" model.

6.1 Approach

The algorithms described in Chapters 3 and 4 were run using training sets from three different sets of data. The Bayesian Dirichlet posterior probability of each resulting network was collected for further analysis. For each network, the log loss score was found using a separate holdout sample from the data. A best so far curve was plotted during each run along with convergence curves for the MCMC algorithms. The Bayesian Dirichlet posterior probabilities and log loss scores were then analyzed using Bayesian procedures to find the 95 percent credible intervals.

6.1.1 Datasets

Results were obtained from the three datasets described below. Two of the three datasets were samples from known networks. Using known networks (i.e. known phenomenon) allowed me to compare the results from the learned networks with the results from the known network. The other dataset is from "real-world" data that has been used in many statistical and graphical model studies. The models induced from this dataset provide a very nice inference problem for any AI system.

6.1.1.1 1X3X3 Network

This dataset is a random sample from a known network, the 1X3X3 network of Figure 5.1. The sample is divided, randomly, into training data and test data. The data consists of seven binary-valued random variables.

The 1X3X3 network was chosen for a couple of reasons. First, when a hidden variable is inserted between the bottom two layers the network is greatly simplified, see Figure 5.2. The number of probabilities required to specify the network reduces from 63 to 21. A modified version, the 3X3 portion of the network in Figure 5.1, of this network was used by Binder et al., and Friedman for learning Bayesian networks with hidden variables. Both approaches were discussed in Chapter 2. Binder et al., [Binder, Koller et al. 1997] used a gradient descent approach to learn the parameters when the hidden variable was manually inserted. Friedman [Friedman 1998] used the SEM algorithm discussed in Chapter 2 to learn where to place the hidden variable. Friedman's approach met with limited success. His algorithm placed a hidden variable in the network but is

was not placed as in Figure 5.2. The second reason for using this network is that the bottom six variables are maximally connected when converted to an undirected graph and triangulated (needed for inference, see Chapter 2). Learning a maximally connected graph can be difficult for a stochastic algorithm because arcs are added and deleted randomly. The probability of generating a maximally connected graph, even with a strong bias to add arcs, is small. Therefore, this network was selected to represent a difficult case for the stochastic algorithms.

6.1.1.2 College Plans

This is a real-world dataset used originally in a study by Sewell and Shah to help make inferences of what influences high school student to attend college [Sewell and Shah 1968]. The dataset consists of 10,318 cases measured from Wisconsin High School students. The variables measured were Sex: male, female; Socioeconomic Status: low, lower middle, upper middle, high; Intelligence Quotient: low, lower middle, upper middle, high; Parental Encouragement: low, high; and College Plans: yes, no. An accurate model of this data makes for a nice inferential task for an intelligent system. One could use it to ask questions such as which variable is a major factor in a student's decision to attend college and for students with low socioeconomic status, can parental encouragement improve a student's chances of attending college. Training data were generated by randomly selecting 25 percent of the cases. The remaining cases were used as test data.

This dataset has been studied extensively using classical and Bayesian techniques, including graphical models, see [Whittaker 1990], [Spirtes, Glymour et al. 1993], and

[Heckerman 1996]. Because it has been studied, models currently exist for comparison purposes. In addition to being a well studied dataset and a nice inferential model, Heckerman suggests there may be a latent variable that helps explain the data. The current agreed upon model has an arc between socioeconomic status and IQ. Intuitively, this appears to be a rather suspicious result. Heckerman suggested removing the arc and adding a hidden variable. From a causal perspective (I'm not advocating causality here, just making an observation) one could interpret the hidden variable as influencing both IQ and socioeconomic status.

6.1.1.3 ALARM Network

The ALARM (A Logical Reduction Mechanism) network is part of a diagnostic system used for monitoring patient ventilation systems in intensive care units. It consists of 37 multi-valued random variables [Beinlich, Suermondt et al. 1989]. Because of its size and complexity it is considered to be the benchmark network for Bayesian network learning algorithms. I generated data from the original network using random sampling. Both the test data and the training data consist of 1000 samples. This is a rather small dataset size for such a large network. I chose this network to demonstrate the algorithms can scale to learn larger networks. I chose the small dataset size to demonstrate the algorithms can generalize well from smaller datasets, even with missing data.

6.2 Stochastic Population-Based Search

The main hypothesis derived from the learning framework states that stochastic population-based search algorithms find networks from incomplete data that perform well

in terms of predictive power. In addition, these algorithms do not suffer the same fate of greedy hill-climbing algorithms of getting "stuck" in the nearest local maximum. I use the three datasets discussed above to illustrate this hypothesis. First, I'll provide results for inducing networks from data with missing values. I'll discuss a couple of interesting findings and show that these algorithms perform better than a simple greedy algorithm. Next, I show that the algorithms are able to learn networks with hidden variables. This is a difficult problem because hidden variables can be induced anywhere in the graph with any number of arcs, but there are only a few node placements that may simplify the network. Finding where to place the hidden variables is the difficult problem. I'll demonstrate using the 1X3X3 and college data that finding hidden variables in graphs is possible, but the results raise more research issues than they resolve.

6.2.1 Missing Data

Figure 6.1 shows the 95 percent credible intervals for the Bayesian Dirichlet and Figure 6.2 shows the 95 percent credible intervals for the log loss for each of the canonical algorithms and the three datasets. These are plots of the Bayesian Dirichlet and log loss for the single best networks found during each run. The EA clearly scores significantly better using the Bayesian Dirichlet. This means that the posterior probability of the single best networks given the training set and sampled missing data is higher for the EA than the MCMC algorithm after 500 generations. Figure 6.3 shows the best so far curves of the EA and MCMC for the 1X3X3 datasets (the best so far curves for the other two datasets look similar). The EA quickly finds higher probable networks
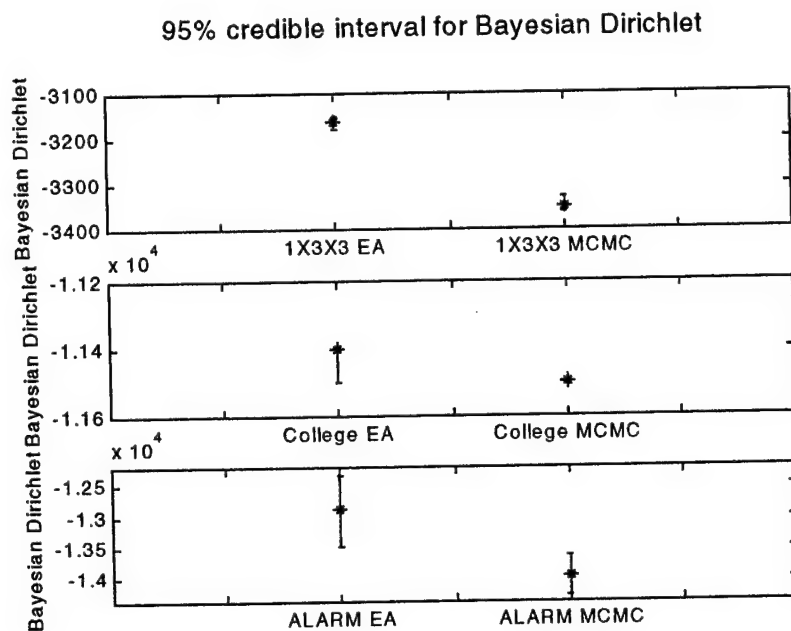
95% credible interval for Bayesian Dirichlet

Figure 6-1 Bayesian Dirichlet for EA and MCMC

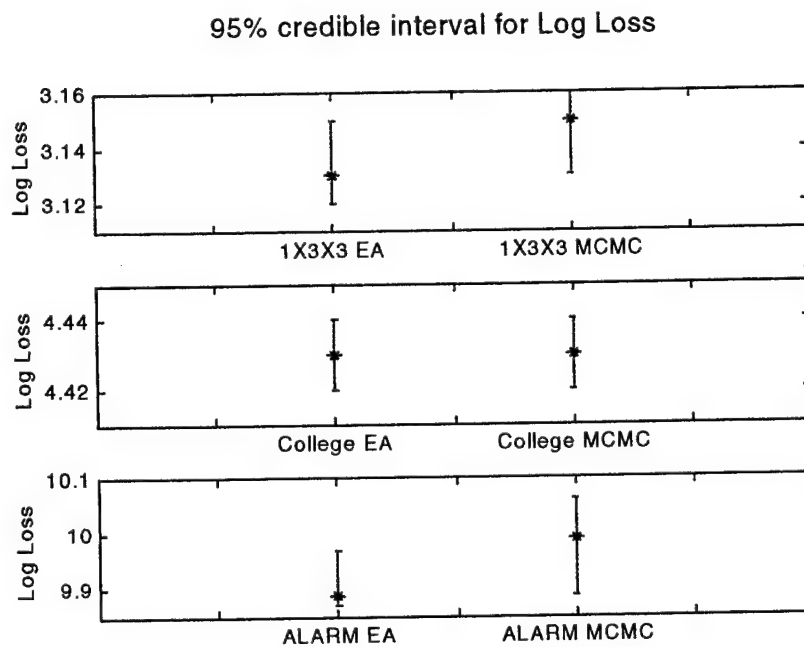95% credible interval for Log Loss

Figure 6-2 Log Loss for EA and MCMC
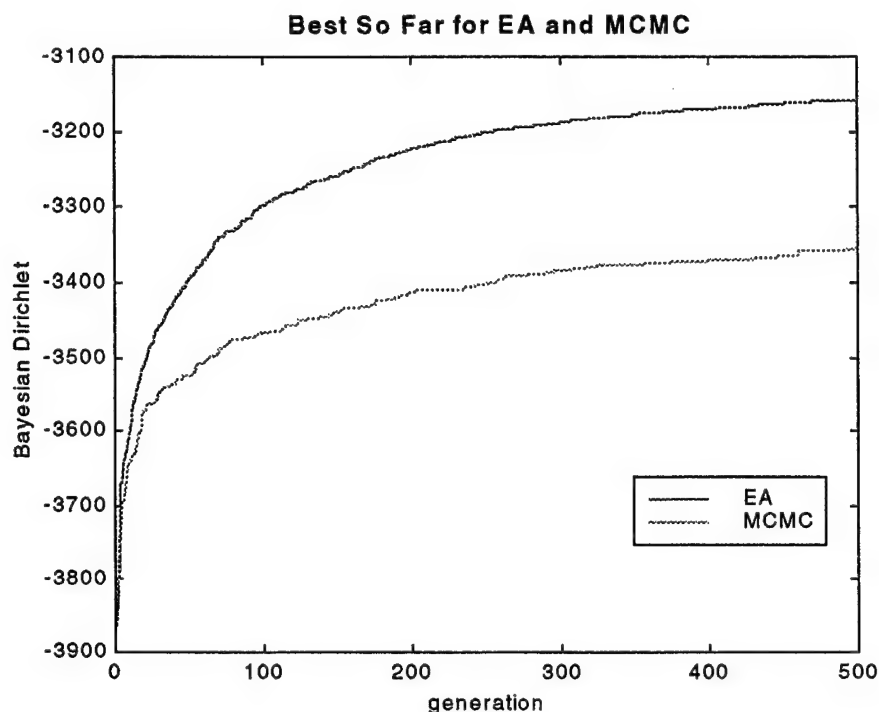
Figure 6-3 Best So Far Curve for EA and MCMC

given the data and sampled missing data than the MCMC algorithm. In fact for the 1X3X3 dataset, the networks found by the EA had higher scores than the original network. How can this be the case? To demonstrate how this is possible a case of flipping a fair coin. A very possible sample from flipping a fair coin 50 times is 30 heads and 20 tails. The phenomenon that produced the data is a binomial distribution, $B(0.5|50,30)$. Now suppose the phenomenon is not known, but the data (samples) are given. Suppose further only two models are under consideration $M_1 = B(0.5|50,30)$ and $M_2 = B(0.6|50,30)$ and they are considered equally likely. Recall from Chapter 2

$$P(M_i|D) = \frac{P(D|M_i)P(M_i)}{P(D)}$$

where M_i is model i = 1,2 and D represents the data. The posterior odds is

$$\frac{P(M_1|D)}{P(M_2|D)} = \frac{P(D|M_1)}{P(D|M_2)} \frac{P(M_1)}{P(M_2)}$$

the prior odds cancels since both models are equally likely, so

$$\frac{P(M_1|D)}{P(M_2|D)} = \frac{B(0.5|50,30)}{B(0.6|50,30)} \frac{0.5}{0.5} \approx 0.36$$

The second model is more than twice as likely than the model that produced the data.

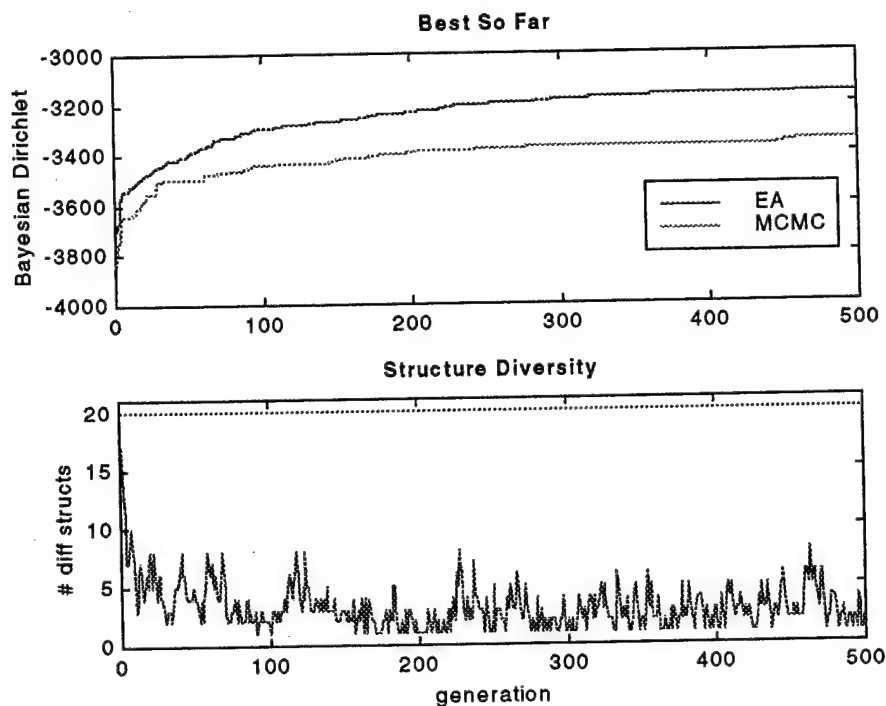**Figure 6-4 Best So Far and Structural Diversity of EA and MCMC**

One can infer from Figure 6.2 that the EA consistently performs better than the MCMC in terms of log loss (predictive capability). However, the 95 percent credible intervals overlap for each of the datasets. This implies that even though the networks found by the MCMC are not as probable as those found by the EA, they are comparable predictive models.

Figure 6.4 provides further insight into the dynamics of both of these algorithms. The top plot compares the best so far curves for an EA run versus a MCMC run. While the bottom plot compares the structure diversity of each algorithm. The structure diversity refers to the number of distinct structures at each generation. Notice that the EA homes in quickly on a few good structures while the MCMC maintains complete structural diversity throughout the run. It should be pointed out that the structures are different for each run providing additional evidence of the multi-modal landscape. The reason for the EA finding more probable networks is as follows. The EA quickly finds a few good structures. The remainder of the run the EA is evolving the missing data to fit the structure. Essentially, the EA is maximizing the likelihood of the data given the structure. Whereas the MCMC is maintaining maximum diversity at the cost of not attempting to quickly maximizing the likelihood score. At this point we can make a couple of observations. If the goal is to find a single most probable model in the fewest generations then use the EA. However, if the goal is to find a set of predictive models that perform comparably well, then use the MCMC.

I also compared the EA and MCMC algorithm with a greedy hill-climbing algorithm. The greedy algorithm is similar to the MCMC in that the next proposed state is found by adding, deleting, or reversing an arc of the current structure. The missing data are also modified in a similar manner as the MCMC. The difference between the greedy approach and MCMC is that the proposed state is accepted only if it scores better than the current state. For comparison purposes the greedy algorithm was run as a

population-based algorithm simulating a single run N times. The results are shown in Figure 6.5. The EA and MCMC algorithms both clearly outperform the greedy hill-climber in terms of finding more probable and better predictive networks.
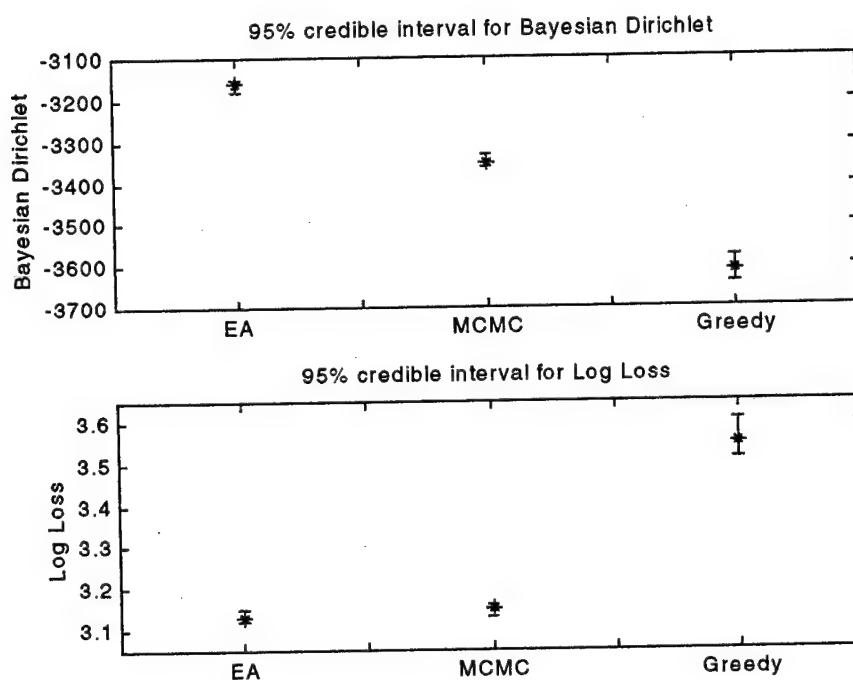
Figure 6-5 Stochastic vs Deterministic

6.2.2 Hidden Variables

From the section above, it is clear that stochastic population-based algorithms find "good" networks from data with missing values. How well do they perform when trying to learn placement of hidden variables? The datasets I used for this set of experiments were the 1X3X3 and college data.

I'll begin with the 1X3X3 data. Recall from Chapter 5, the hidden variable placed as in Figure 5.2 reduces the number of parameters needed to specify the model by more than

50 percent. I added a random variable to the dataset by adding a column with all missing

values. Then I ran the EA and MCMC algorithms with two modifications. First, the

initial population was generated by starting with the hidden variable as the root node with

arcs to the remaining variables and then randomly placing arcs between the remaining

variables. This represents the prior that a hidden variable exists. The second

modification was to add a mutation operator that added an arc between the hidden

variable and another variable in the network. This bias is necessary because the initial

population of missing data is randomly generated. Without this bias both the EA and

MCMC will quickly rule out any structures with arcs to or from the hidden variable. The

bias allows the missing data to evolve toward a correlation between the hidden variable

and other variables in the network.

Original   EA   MCMC
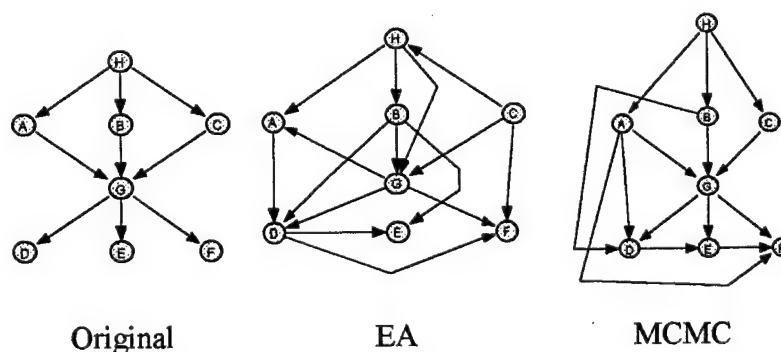
Figure 6-6 Hidden Variable for 1X3X3

A couple of representative networks induced by the EA and MCMC are shown in

Figure 6.6. As can be seen, the networks found by the EA and MCMC are fairly close to

the expected network. The induced networks have too many additional arcs. This may

be due to the strong bias to add arcs to the hidden variable. Notice the network found by

the MCMC had the correct arc placements but had four additional arcs. For comparison purposes the 95 percent credible interval for log loss of the networks found by the EA was [3.13 3.16] and the interval for the MCMC was [3.12 3.16]. These intervals overlap those found for missing values.

Heckerman   EA   MCMC
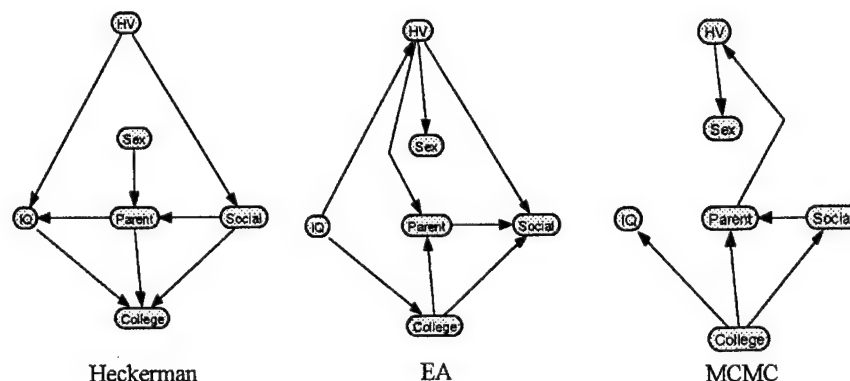
Figure 6-7 Hidden Variable for College Data

Figure 6.7 shows representative networks found from the college data using the EA and MCMC algorithms. Also shown in Figure 6.7 is the network proposed by Heckerman [Heckerman 1996]. The Bayesian Dirichlet and log loss scores for each of these networks were very close, with the 95 percent credible intervals overlapping. From an explanatory perspective, each network is reasonable. Recall the original network had an arc between socioeconomic status and IQ, Heckerman's proposed network removed that arc and added a hidden variable to explain the relationship between the two variables. The network found by the EA can be interpreted similarly with an additional twist. The hidden variable also explains the relationship between parental encouragement and the child's sex. The network learned by the MCMC did not find a relationship

between socioeconomic status and IQ strong enough to warrant an arc. It did remove the arc between sex and parental encouragement and added a hidden variable.

Hidden variables (also known as latent variables) have been used for years in graphical models, see [Cheeseman and Stutz 1995] and [Bishop 1999]. The typical solution is to manually add the hidden variable. As demonstrated above we are closer to being able to automatically place hidden variables in graphical models. The only other attempt to learn hidden variables was by Friedman [Friedman 1998]. His results were similar to mine. Both approaches use a rather blind search. Clearly a more principled approach is required. One possibility is to use domain knowledge to guide the search process.

6.3   Global Information Exchange Speeds Convergence in MCMC

As mentioned previously, the MCMC algorithms can be very slow converging to the stationary distribution. In Chapters 3 and 4 I discussed some of the efforts researchers have taken to overcome this problem. The second hypothesis proposed in Chapter 1 was that using global information to propose new states will speed up convergence. To demonstrate this I ran the canonical EMCMC and adaptive mutation MCMC proposed in Chapter 4 on the 1X3X3 dataset.

Figure 6.8 shows the convergence curves for the MCMC, canonical EMCMC, and the MCMC with adaptive mutation for structure. The curves show the convergence limits of 1.1 and 1.2 as suggested by Gelman [Gelman and Rubin 1992]. The canonical EMCMC did not show any real improvement over the MCMC algorithm. However, the adaptive mutation operator doubled the convergence rate of the MCMC. The results for
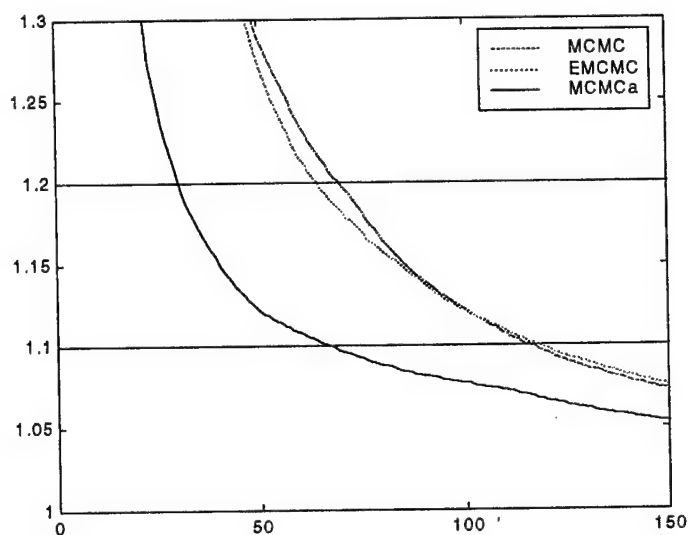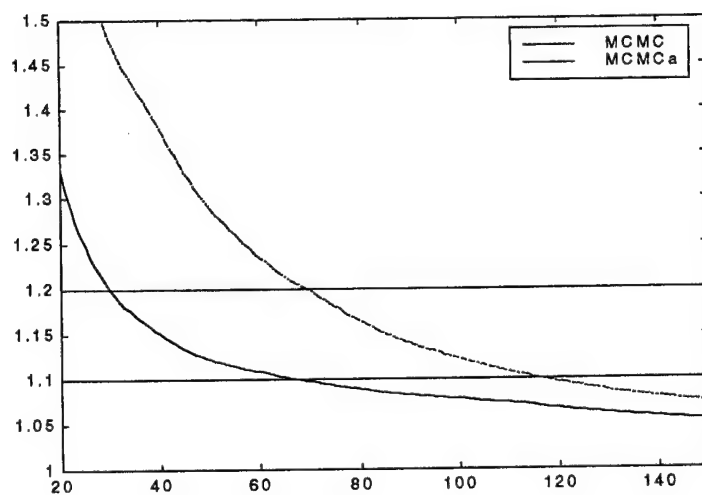
Figure 6-8 Convergence Curves for 1X3X3

Figure 6-9 Convergence Curves for College Data

the adaptive mutation are very encouraging. To provide more evidence in support of the

adaptive mutation operator, I ran MCMC with adaptive mutation on the college dataset.

The results are shown in Figure 6.9. Once again the adaptive mutation operator doubled the convergence rate. These results demonstrate that taking advantage of global information can increase convergence rates of MCMC algorithms.

The results from the canonical EMCMC were disappointing. It seemed reasonable that crossover will increase the distribution of higher probable networks, thus increasing convergence. After further thought, I believe the problem with my approach is that I selected parents randomly and not by fitness. An EA selects parents based on fitness and then creates offspring by crossover and mutation. Crossover works in this case because the algorithm is exchanging information between highly fit individuals. When the EMCMC randomly selects parents, there is just as good of a chance of selecting less fit individuals to crossover. I attempted a modification where I ranked the individuals by fitness and performed crossover based on their rank. In other words, higher fit individuals were given a higher probability to crossover. This approach did not improve convergence. Again, I believe the problem is due to the fact that the less fit individuals are still in the population. As the higher fit couples produce higher fit offspring, the lower fit couples still produce low fit offspring. Only mutation allows these individuals to evolve to higher fit individuals. Another interesting research topic is to find a way to use an EA selection scheme while maintaining detailed balance. This may require a way to allow the population to increase and decrease.

The results from the adaptive mutation operator for structure indicate that this operator can be used to improve mixing, thus almost doubling the rate of convergence. These results are encouraging. The logical follow-on question is how much improvement

can be gained by adding an adaptive mutation operator for the missing data? I

implemented an adaptive operator for the missing data and the results were quite

significant. Figure 6.10 compares the MCMC algorithm with the MCMC with adaptive

mutation for structure and missing data. The MCMC with adaptive mutation found

overall more probable networks. This means the previous results from the canonical

MCMC algorithm were not from the stationary distribution because the MCMC had

never converged. Figure 6.10 in fact shows that even after 3000 generations the MCMC

still has not converged. I ran the MCMC for 5000 iterations without convergence. This

means the MCMC with adaptive mutation over structure and missing data increases

convergence by greater than a factor of ten.
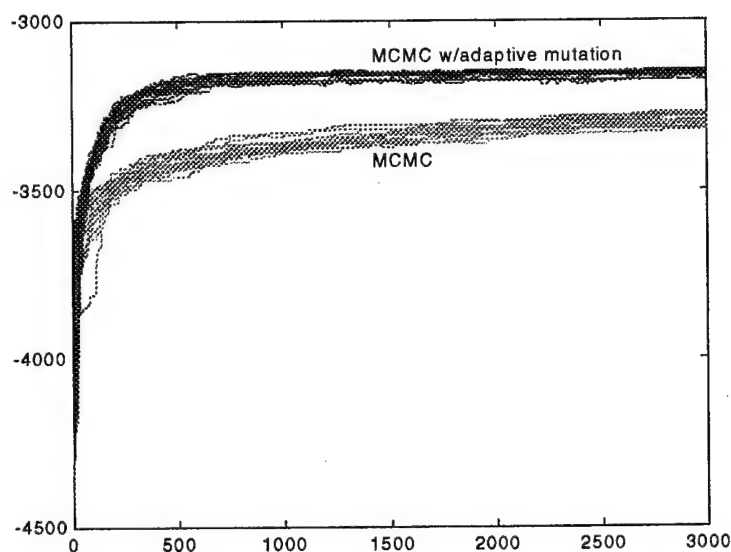
Figure 6-10 Markov Chains for MCMC and MCMC with adaptive mutation

So why did the convergence charts in Figures 6.8 and 6.9 show the MCMC had

converged? Convergence of MCMC chains to the stationary distribution is very difficult

to monitor. Most approaches either rely on plotting the chains and visually monitoring for convergence or some measure such as the scale reduction mentioned in Chapter 5 or some measure of autocorrelation. Without first seeing the results from the MCMC with adaptive mutation the graphical approach would have predicted convergence because the "true" convergence is so slow a trend toward the "true" stationary distribution is difficult to observe. Likewise the metric indicates convergence because the within and between variance are almost equal. Once the MCMC with adaptive mutation results are observed it is clear the canonical MCMC chains have not converged.
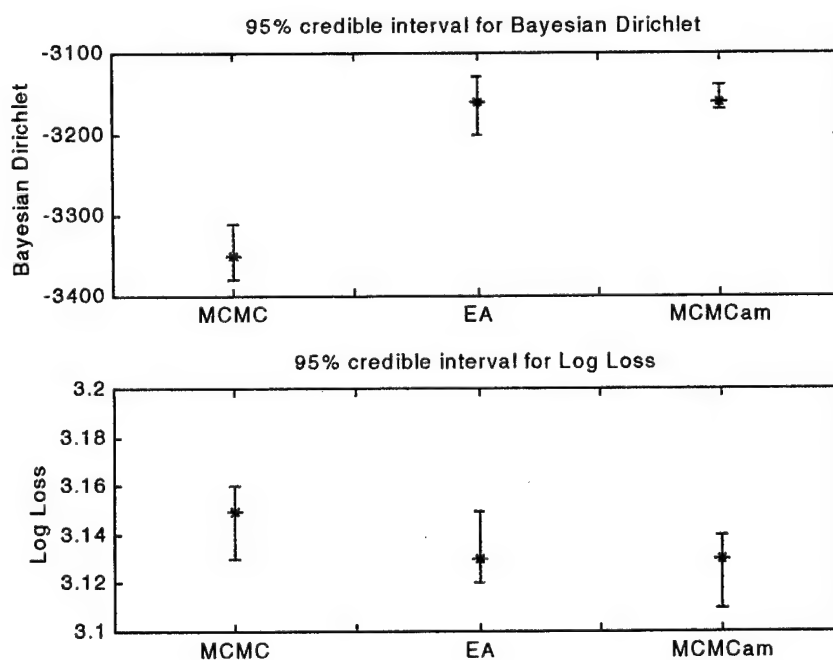
Figure 6-11 MCMC with adaptive mutation vs EA and MCMC

Figure 6.11 shows that the MCMC with adaptive mutation not only finds better networks than the canonical MCMC it finds networks that are as good or arguably even better than the EA. The MCMC with adaptive mutation finds networks that are just as

probable as the EA with a slight improvement over the EA in log loss. Figure 6.12 shows the results of the Best So Far curves and diversity plots for the EA and MCMC with adaptive mutation. The MCMC algorithm finds probable networks just as quickly as the EA while maintaining maximum structural diversity.
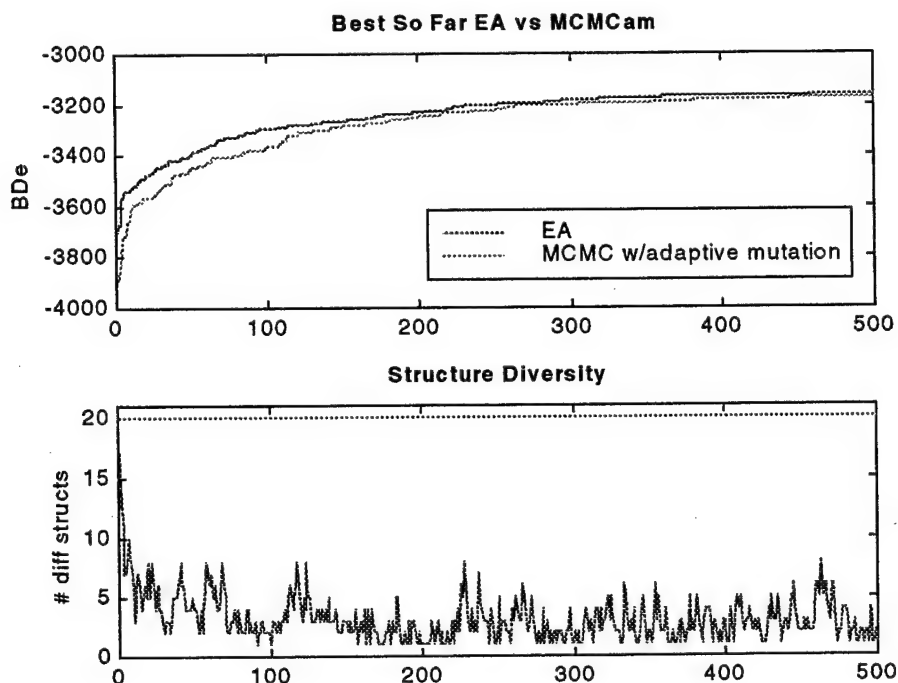
Figure 6-12 Best So Far and Diversity of MCMC with adaptive mutaiton and EA

6.4 Multiple Models versus Single "Best" Model

The final hypothesis proposed in Chapter 1 was that multiple models have higher predictive power than the single "best" model. This seems reasonable since the data are stochastic, so there is no reason to believe the most probable model represents the data. Recall the coin flipping example in Section 6.2. The data suggested model 2 was more probable than the model that produced the data. In addition, empirical results from

Madigan et al., suggests this hypothesis may be true [Madigan and Raftery 1994]. To

demonstrate this hypothesis I ran the EA, MCMC, and MCMC with adaptive mutation

algorithms on the 1X3X3 dataset and took the final 20 individuals as a sample of multiple

models. I used (5.16) to calculate the log loss score for the multiple models from each

run. Since the 20 models from both the MCMC runs were independent samples from the

stationary distribution (actually only the MCMC with adaptive mutation were from the

stationary distribution as the MCMC had not converged), I set the priors of the models

equal. I did the same for the EA, though there is no theoretical or practical reason to do

so since the samples can not be shown to be from the stationary distribution. In fact, they

are from a mode-biased distribution. My logic for setting the models from the EA to

equal priors was for comparison purposes.

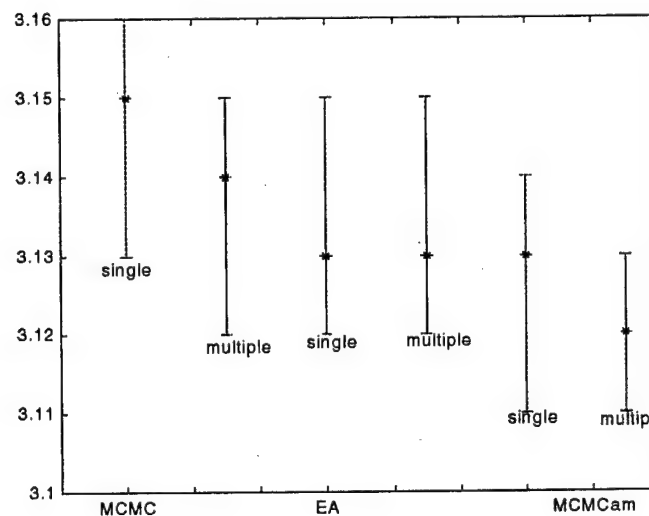**Figure 6-13 Multiple Models vs Single "Best" Model**

Figure 6.13 shows the 95 percent credible intervals for the log loss scores of the multiple models and the single "best" model from each algorithm. The multiple model results for both the MCMC and MCMC with adaptive mutation were slightly better than the single model results. Even though the credible intervals overlap, when I combine these results with the prior results from Madigan, the evidence further supports the hypothesis. There was no performance increase for the EA because the multiple model approach was essentially averaging over the same structure since there are only a few unique structures in the sample.

Even though the additional computation required performing model averaging is not significant, one may wonder why commit the additional resources for such small gains? If the additional predictive power is not needed, one may want to only use the single "best" model found. This works well if the goal is strictly prediction. However, if the single model were used in any other way such as making inferences about the phenomenon that produced the data, the single model will be incorrect. Multiple models sampled from a stationary distribution should allow us to make structural inferences based on arc distributions. Another advantage for using multiple models is when the phenomenon is known to vary over time. The structural diversity of samples from an MCMC may very well capture the changes much better than a single model.

6.5 Summary

This chapter demonstrated the hypotheses articulated in Chapter 1. I've shown that stochastic population-based algorithms find "good" predictive Bayesian networks from incomplete data. I've demonstrated this for data with missing values and hidden

variables. I've also demonstrated that global information can be used in proposing new states that speed convergence to the stationary distribution for MCMC algorithms. Finally, I've provided additional evidence that multiple models have higher predictive ability than the single "best" model.

Chapter 7 CONCLUSIONS AND FUTURE DIRECTION

7.1 Conclusion

Learning in intelligent systems is a field of study that draws techniques from a wide variety of disciplines. In this dissertation alone, I used techniques and ideas from artificial intelligence, probability theory, evolutionary biology, statistical mechanics, and information theory. These are only a few of the disciplines claiming a stake in learning. Because of this diversity, there is little agreement on a learning framework. Without a framework, the field lacks direction. With a unified learning framework, researchers will be able to tell where the field has been, its current state, and its direction for the future.

I have developed and presented a framework that encompasses the current state-of-the-art of machine learning. From the framework I identified three hypotheses that apply to a currently active area in machine learning: inducing Bayesian networks from incomplete data. From these hypotheses I developed algorithms that advanced the state-of-the-art of learning Bayesian networks specifically and Markov Chain Monte Carlo algorithms in general. I demonstrated that stochastic population-based algorithms, namely EA and MCMC, find highly probable networks from incomplete data and these networks have good predictive power. Further, I demonstrated that global information can be used to propose new states that speed up convergence in MCMC algorithms.

Finally, I provided additional evidence that multiple model averaging outperforms single "best" networks.

7.2 Future Direction

Though this dissertation has demonstrated several advances in the field of machine learning, it has also identified several areas for future research.

1. Learning Framework

- The framework, as defined in Chapter 1, is a top-level framework that allows researchers to identify hypotheses for future research. It is a good start but is not fully developed. I believe a model developed to the second or third level of the object model will be even more helpful to defining the future direction of the field.

- An interesting research area is to treat the framework as a representation within the framework and develop a meta-learning algorithm for inducing learning algorithms.

- From the framework, it should now be possible to identify combinations of components that work well and combinations that do not work well. For instance, what combination of components of the framework work well on categorical observations or time-varying observations.

- The framework should pave the way for researchers to explore further methods of going from one representation to another depending on the learning environment. For example, if we know that a rule-based representation works well with a certain combination of components and the current representation is a Bayesian

network, it would be helpful to be able to move from a Bayesian network to a rule-based representation. In addition, it may be that some types of inference are better suited for classification trees and not neural networks. It would be nice to have a method that converts the representation from classification trees to neural networks. It seems reasonable to be able to switch from one representation to another, humans do it all the time.

- We need to develop principled approaches to inquiry in learning systems. This will allow us to develop discovery learning systems and take us a step closer to intelligent systems. Only through inquiry can we develop hypotheses that add truly new knowledge.

2. Stochastic population-based algorithms

- I demonstrated how the EA converges quickly to a few structures. At that point, exploration over structures essentially stops. Further investigation is required to find ways to keep exploration active while maintaining strong exploitation. One approach that has been explored in genetic algorithms is using speciation [Spears 1994]. One could add tag bits to individuals in the population. The tags identify different species. Only individuals of the same species are allowed to breed. The species will act as separate populations converging to possibly different modes. A small mutation rate is applied to the species tag to allow individuals to migrate to another species over time. The new individual will be different enough from the individuals of the current species to cause a disturbance in the distribution and possibly cause the species to converge on a different mode.

- Another possible improvement to the EA is to combine the structure search with Friedman's SEM algorithm. Instead of a second population of missing data values, there is only one population of structures. The SEM is called for each structure and returns with possibly a separate structure. Such Lamarkian evolution may prove to find even better performing networks by finding a better set of parameters since imputation methods are known to underestimate variance. This approach is known also known as a hybrid evolutionary algorithm.

- Another possible improvement is to first run an EA, possibly with speciation, to generate the initial population. Then run an MCMC algorithm. This possibly will speed convergence and may in fact find solutions with higher posterior probabiltiy.

3. Global Information Exchange

- The canonical EMCMC did not speed convergence as expected. I believe the reason for this is due to the random selection of parents. A possible solution to this is to use selection schemes such as those used in evolutionary algorithms. Unfortunately, these selection scheme may select some individuals (more fit) more than once for breeding and may not select others individuals (less fit) at all. This causes a problem for detailed balance. It should be possible to develop a meta-MCMC that allows for this kind of selection. One possibility is an MCMC that allows the chains to die and multiple. Proving detailed balance for such an MCMC can be very difficult.

- The adaptive mutation operator performed very well in the Bayesian network domain. Adaptive mutation operators for other domains need to be developed and tested.

4. Multiple Models versus Single "Best" Model

- Further research needs to be conducted to find when multiple model averaging performs better than the single "best" model and vice versa.

- An interesting research topic would be to identify a problem where observations are given over time. I believe that at the beginning of this learning process the learner should keep several probable models to represent the phenomenon. As time passes and more observations are made, some models will become more probable while others less probable and drop out as credible. This scenario is probably a good representation of one way intelligent systems learn.

- Along the same lines, another hypothesis may be that when observing time-varying phenomena is it better for the learner to carry multiple models as opposed to a single model. For instance, the intelligent system may have learned to survive in an environment where energy is plentiful and predators are few. Over time the environment may change to a situation where food is scarce and predators abound. A learner that carries multiple models may find a few models that work well in the new environment.

BIBLIOGRAPHY

AAAI98 (1998). . Proceedings of the Fifteenth National Conference on Artificial Intelligence, Madison, WI, AAAI Press/The MIT Press.

Abe, N., H. Li, et al. (1995). On-Line Learning of Binary Lexical Relations Using Two-dimensional Weighted Majority Algorithms. The XII International Conference on Machine Learning, Tahoe City, CA, Morgan Kauffmann Publishers.

Almueallim, H., Y. Akiba, et al. (1995). On Handling Tree-Structured Attributes in Decision Tree Learning. The XII International Conference on Machine Learning, Tahoe City, CA, Morgan Kauffmann Publishers.

Auer, P., R. C. Holte, et al. (1995). Theory and Application of Agnostic PAC-Learning with Small Decision Trees. The XII International Conference on Machine Learning, Tahoe City, CA, Morgan Kauffmann Publishers.

Back, T. (1996). Evolutionary Algorithms in Theory and Practice. New York, Oxford University Press.

Bain, L. and M. Engelhardt (1992). Introduction to Probability and Mathematical Statistics. Belmont, CA, Duxbury Press.

Baker, J. E. (1985). Adaptive Selection Methods for Genetic Algorithms. Proceedings of the 1st International Conference on Genetic Algorithms and Their Applications, Lawrence Erlbaum Associates.

Baluja, S. and R. Caruana (1995). Removing the Genetics from the Standard Genetic Algorithm. The XII International Conference on Machine Learning, Tahoe City, CA, Morgan Kauffmann Publishers.

Beinlich, I. A., H. J. Suermondt, et al. (1989). The ALARM Monitoring System: A Case Study with Two Probabilistic Inference Techniques for Belief Networks. Proceedings of the Second European Conference on Artificial Intelligence in Medicine.

Bernardo, J. M. and A. F. M. Smith (1994). Bayesian Theory. Chichester, John Wiley & Sons.

Beveridge, J. R. (1998). Optimal 2D Model Matching Using a Messy Genetic Algorithm. Proceedings of the Fifteenth National Conference on Artificial Intelligence, Madison, WI, AAAI Press/The MIT Press.

Binder, J., D. Koller, et al. (1997). "Adaptive Probabilistic Networks with Hidden Variables." Machine Learning.

Bishop, C. M. (1999). Latent Variable Models. Learning in Graphical Models. Cambridge, MA, The MIT Press: 371-403.

Brodie, M. and G. DeJong (1998). Iterated Phantom Induction: A Little Knowledge Can Go A Long Way. Proceedings of the Fifteenth National Conference on Artificial Intelligence, Madison, WI, AAAI Press/The MIT Press.

Cheeseman, P. and J. Stutz (1995). Bayesian Classification (AutoClass): Theory and Results. Advances in Knowledge Discovery and Data Mining. U. Fayyad, G. Piatesky-Shapiro, P. Smyth and R. Uthurusamy. Menlo Park, AAAI Press: 153-180.

Chickering, D. M. (1996). Learning Bayesian Networks from Data. Computer Science. Los Angeles, University of California Los Angeles: 231.

Cichosz, P. and J. J. Mulawka (1995). Fast and Efficient Reinforcement Learning with Truncated Temporal Differences. The XII International Conference on Machine Learning, Tahoe City, CA, Morgan Kauffmann Publishers.

Cooper, G. F. (1995). "A Bayesian Method for Learning Belief Networks that Contain Hidden Variables." Journal of Intelligent Information Systems 4: 71-88.

Cooper, G. F. and E. Herskovits (1992). "A Bayesian Method for the Induction of Probabilistic Networks from Data." Machine Learning 9: 309-347.

Davis, T. E. and J. C. Principe (1991). A Simulated Annealing like Convergence Theory for the Simple Genetic Algorithm. Proceedings of the 4th International Conference on Genetic Algorithms, San Diego.

Dawid, A. P. (1979). "Conditional Independence in Statistical Theory." Journal of the Royal Statistical Society 41: 1-31.

Dawid, A. P. (1980). "Conditional Independence for Statistical Operations." Annals of Statistics 8: 598-617.

Dawid, A. P. (1982). "The Well Calibrated Bayesian." Journal of the American Statistical Association 77: 605-613.

Dawid, A. P. (1984). "Statistical Theory, the Prequential Approach." Journal of the Royal Statistical Society A **147**: 278-292.

De Finetti, B. (1995). Theory of Probability. Chichester, John Wiley & Sons.

Deb, K. and M. Goyal (1997). Optimizing Engineering Designs using a Combined Genetic Search. Proceedings of the Seventh International Conference on Genetic Algorithms, East Lansing, MI, Morgan Kauffmann Publishers.

DeGroot, M. H. (1970). Optimal Statistical Decisions. New York, McGraw-Hill Book Company.

DeJong, K. A. (1975). An Analysis of the Behavior of a Class of Genetic Adaptive Systems. Computer and Communication Sciences. Ann Arbor, MI, University of Michigan: 256.

DeJong, K. A. (1993). Genetic Algorithms are NOT Function Optimizers. Foundations of Genetic Algorithsm. L. D. Whitley. San Mateo, CA, Morgan Kaufmann Publishers. **2**: 5-17.

DeJong, K. A. and W. M. Spears (1990). An Analysis of the Interacting Roles of Population Size and Crossover in Genetic Algorithms. Proceedings of the First International Conference on Parallel Problem Solving from Nature, Dortmund, Germany.

DeJong, K. A., W. M. Spears, et al. (1995). Using Markov Chains to Analyze GAFOs. Foundations of Genetic Algorithms. L. D. Whitley and M. D. Vose. San Mateo, CA, Morgan Kaufmann. **3**: 115-137.

Dempster, A. P., N. M. Laird, et al. (1977). "Maximum Likelihood Estimation from Incomplete Data via the EM Algorithm." Journal of the Royal Statistical Society **V39**: 1-38.

Duda, R. O. and P. E. Hart (1973). Pattern Classification and Scene Analysis. New York, John Wiley & Sons.

Eschelman, L., R. Caruana, et al. (1989). Biases in the Crossover Landscape. Proceedings of the 3rd International Conference on Genetic Algorithms, Morgan Kaufman Publishing. Feller, W. (1968). An Introduction to Probability Theory and Its Applications. New York, John Wiley.

Fogel, L. J. (1991). System Identification through Simulated Evolution: A Machine Learning Approach to Modeling. Needham Heights, Ginn Press.

Friedman, N. (1998). The Bayesian Structural EM Algorithm. Proceedings of the Fourteenth Conference on Uncertainty in Artificial Intelligence, Madison, WI, Morgan Kaufmann Publishers.

Friedman, N. (1998). Learning Belief Networks in the Presence of Missing Values and Hidden Variables. **Fourteenth International Conference on Machine Learning (ICML-97)**, Vanderbilt University, Morgan Kaufmann Publishers.

Fukunaga, K. (1990). Introduction to Statistical Pattern Recognition. New York, Academic Press.

Fung, R. M. and S. L. Crawford (1990). A System for Induction of Probabilistic Models. Eighth National Conference on Aritificial Intelligence, Boston, AAAI Press.

Geiger, D., D. Heckerman, et al. (1996). Asymptotic Model Selection for Directed Networks with Hidden Variables. Redmond, WA, Microsoft Research.

Gelman, A. and D. B. Rubin (1992). "Inference from Iterative Simulation using Multiple Sequences." Statistical Science 7: 457-472.

Geman, S. and D. Geman (1984). "Stochastic Relaxation, Gibbs Distributions, and the Bayesian Restoration of Images." IEEE Transaction on Pattern Analysis and Machine Intelligence 6(6): 721-741.

Gennari, J. H., P. Langley, et al. (1989). "Models of Incremental Concept Formation." Artificial Intelligence 40(1): 11-61.

Gilks, W. R., S. Richardson, et al. (1996). Markov Chain Monte Carlo in Practice. London, Chapman & Hall.

Gilks, W. R. and G. O. Roberts (1996). Strategies for Improving MCMC. Markov Chain Monte Carlo in Practice. W. R. Gilks, S. Richardson and D. J. Spiegelhalter. London, Chapman & Hall: 89-114.

Ginsberg, A., S. M. Weiss, et al. (1988). "Automatic Knowledge Base Refinement for Classification Systems." Artificial Intelligence 35(2): 197-226.

Goldberg, D. E. (1989). Genetic Algorithms in Search, Optimization, and Machine Learning. Reading, Addison-Wesley Publishing Company, Inc.

Goldberg, D. E. and K. Deb (1991). A Comparative Analysis of Selection Schemes Used in Genetic Algorithms. Foundations of Genetic Algorithms. G. J. E. Rawlins. San Mateo, CA, Morgan Kaufmann Publishers. 1: 69-93.

Goodman, I. R., H. T. Nguyen, et al. (1991). Conditional Inference and Logic for Intelligent Systems: A Theory of Measure-Free Conditioning. Amsterdam, North-Holland.

GP98 (1998). . Proceedings of the Third Annual Genetic Programming Conference, Madison, WI, Morgan Kaufmann.

Grove, A. J. and D. Schuurmans (1998). Boosting in the limit: Maximizing the Margin of Learned Ensembles. Proceedings of the Fifteenth National Conference on Artificial Intelligence, Madison, WI, AAAI Press/The MIT Press.

Hastings, W. K. (1970). "Monte Carlo Sampling Methods using Markov Chains and their Applications." Biometrika 57(1): 97-109.

Heckerman, D. (1996). A Tutorial on Learning with Bayesian Networks. Redmond WA, Microsoft.

Heckerman, D., D. Geiger, et al. (1995). "Learning Bayesian Networks: The Combination of Knowledge and Statistical Data." Machine Learning 20: 197-243.

Holland, J. H. (1995). Adaptation in Natural and Artificial Systems. Cambridge, MIT Press.

Holmes, C. C. and B. K. Mallick (1998). Parallel Markov Chain Monte Carlo Sampling: An Evolutionary Based Approach. London, Imperial College.

Jeffreys, S. H. (1998). Theory of Probability. Oxford, Clarendon Press.

Jensen, F. V. (1996). An Introduction to Bayesian Networks. New York, Springer. Jenzarli, A. (1996). Solving Influence Diagrams using Gibbs Sampling. Learning from Data. D. Fisher and H. J. Lenz. New York, Springer. 112: 59-68.

Jordan, M. I., Ed. (1999). Learning in Graphical Models. Cambridge, MA, The MIT Press.

Kearns, M., Y. Mansour, et al. (1999). An Information-Theoretic Analysis of Hard and Soft Assignment Methods for Clustering. Learning in Graphical Models. M. I. Jordan. Cambridge, MA, The MIT Press: 495-520.

Kirkpatrick, S., C. D. Gelatt, et al. (1983). "Optimization by Simulated Annealing." Science 220: 671-680.

Kitano, H. (1990). "Designing Nerual Networks using Genetic Algorithms with Graph Generation Systems." Complex Systems 4: 461-476.

Kolmogorov, A. N. (1956). Foundations of the Theory of Probability (1933). New York, Chelsea Publishing.

Kubat, M., I. Bratko, et al. (1998). A Review of Machine Learning Methods. Machine Learning and Data Mining: Methods and Applications. R. S. Michalski, I. Bratko and M. Kubat. Chichester, John Wiley & Sons: 3-69.

Larranaga, P., R. Murga, et al. (1996). Structure Learning of Bayesian Networks by Hybrid Genetic Algorithms. Learning from Data: Artificial Intelligence and Statistics V. D. Fisher and H.-J. Lenz. New York, Springer: 450.

Larranaga, P., M. Poza, et al. (1996). "Structure Learning of Bayesian Networks by Genetic Algorithms: A Performance Analysis of Control Parameters." IEEE Journal on Pattern Analysis and Machine Intelligence 18(9): 912-926.

Laskey, K. B. (1999). Personal Communication.

Lauritzen, S. (1996). Graphical Models. Oxford, Oxford Science Publications.

Lauritzen, S. L. (1995). "The EM algorithm for graphical association models with missing data." Computational Statistics & Data Analysis 19: 191-201.

Little, R. and D. Rubin (1987). Statistical Analysis with Missing Data. New York, John WIley & Sons.

Madigan, D. and A. E. Raftery (1994). "Model Selection and Accounting for Model Uncertainty in Graphical Models using Occam's Window." Journal of the American Statistical Association 89: 1335-1346.

Madigan, D., A. E. Raftery, et al. (1996). Bayesian Model Averaging. AAAI Workshop on Integrating Multiple Learned Models.

Madigan, D., A. E. Raftery, et al. (1994). Strategies for Graphical Model Selection. Selecting Models from Data: Artificial Intelligence and Statistics IV. P. Cheeseman and W. Oldford, Springer Verlag: 91-100.

Madigan, D. and J. York (1995). "Bayesian Graphical Models for Discrete Data." International Statistical Review 63: 215-232.

Merz, P. and B. Freisleben (1997). A Genetic Local Search Approach to the Quadratic Assignment Problem. Proceedings of the Seventh International Conference on Genetic Algorithms, East Lansing, MI, Morgan Kauffmann Publishers.

Metropolis, N., A. W. Rosenbluth, et al. (1953). "Equations of State Calculation by Fast Computing Machines." Journal of Chemical Physics **21**: 1087-1092.

Michalski, R. (1994). Inferential Theory of Learning: Developing Foundations for Multistrategy Learning. Machine Learning: A Multistrategy Approach. R. Michalski and G. Tecuci. San Francisco, Morgan Kaufman Publishers. **4:** 782.

Michalski, R. and G. Tecuci, Eds. (1994). Machine Learning: A Multistrategy Approach. San Francisco, Morgan Kaufmann Publishers.

Michie, D., D. J. Spiegelhalter, et al. (1994). Machine Learning, Neural and Statistical Classification. New York, Ellis Norwood.

Mitchell, T. M. (1997). Machine Learning. Boston, McGraw-Hill.

Mitchell, T. M., P. E. Utgoff, et al. (1993). Learning by Experimentation: Acquiring and Refining Problem-Solving Heuristics. Readings in Knowledge Acquisition and Learning. B. G. Buchanan and D. C. Wilkins. San Mateo, CA, Morgan Kauffmann Publishers: 504-517.

Mooney, R., J. (1993). Explanation Generalization in EGGS. Readings in Knowledge Acquisition and Learning. B. G. Buchanan and D. C. WIlkins. San Mateo, CA, Morgan Kauffmann Publishers: 556-575.

Nagata, Y. and S. Kobayashi (1997). Edge Assembly Crossover: A High-power Genetic Algorithm for Traveling Salesman Problem. Proceedings of the Seventh International Conference on Genetic Algorithms, East Lansing, MI, Morgan Kauffmann Publishers.

Neal, R. M. (1993). Probabilistic Inference Using Markov Chain Monte Carlo Methods. Toronto, University of Toronto.

Neapolitan, R. E. (1989). Probabilistic Reasoning in Expert Systems: Theory and Algorithms. New York, John Wiley & Sons, Inc.

Newell, A. and H. A. Simon (1995). Computer Science as Empirical Inquiry. Computation & Intelligence. G. F. Luger. Menlo Park, AAAI Press/The MIT Press: 91-119.

Nilsson, N. J. (1998). Artificial Intelligence: A New Synthesis. San Francisco, Morgan Kaufmann Publishers, Inc.

Nix, A. E. and M. D. Vose (1992). "Modelling Genetic Algorithms with Markov Chains." Annals of Mathematics and Artificial Intelligence **5**: 79-88.

O'Hagan, A. (1994). <u>Bayesian Inference</u>. London, Edward Arnold.

Pearl, J. (1988). <u>Probabilistic Reasoning in Intelligent Systems: Networks of Plausible Inference</u>. San Francisco, Morgan Kaufmann Publishers, Inc.

Press, S. J. (1989). <u>Bayesian Statistics: Principles, Models, and Applications</u>. New York, John Wiley & Sons.

Quinlan, J. R. (1986). "Induction of Decision Trees." <u>Machine Learning</u> 1: 81-106.

Robinson, R. W. (1977). Counting Unlabeled Acyclic Digraphs. <u>Lecture Notes in Mathematics, 622: Combinatorial Mathematics V</u>. C. H. C. Little. New York, Springer-Verlag: 622.

Rubin, D. B. (1987). <u>Multiple Imputation for Nonresponse in Surveys</u>. New York, Wiley.

Rudolph, G. (1994). "Massively Parallel Simulated Annealing and its relation to Evolutionary Algorithms." <u>Evolutionary Computation</u> 1(4).

Rumbaugh, J., M. Blaha, et al. (1991). <u>Object-Oriented Modeling and Design</u>. Englewood City, NJ, Prentice Hall.

Russell, S. and P. Norvig (1995). <u>Artificial Intelligence: A Modern Approach</u>. Upper Saddle River, NJ, Prentice Hall.

Sarma, J. and K. DeJong (1997). <u>An Analysis of Local Selection Algorithms in a Spatially Structured Evolutionary Algorithm</u>. Proceedings of the Seventh International Conference on Genetic Algorithms, East Lansing, MI, Morgan Kauffmann Publishers.

Saul, L. and M. Jordan (1999). A Mean Field Learning Algorithm for Unsupervised Neural Networks. <u>Learning in Graphical Models</u>. M. I. Jordan. Cambridge, MA, The MIT Press: 541-554.

Savage, L. J. (1972). <u>The Foundations of Statistics</u>. New York, Dover Publications, Inc.

Schoenauer, M. and Z. Michalewicz (1997). <u>Boundary Operators for Constrained Parameter Optimization Problems</u>. Proceedings of the Seventh International Conference on Genetic Algorithms, East Lansing, MI, Morgan Kauffmann Publishers.

Schum, D. A. (1994). <u>Evidential Foundations of Probabilistic Reasoning</u>. New York, John Wiley & Sons, Inc.

Schwefel, H.-P. (1995). <u>Evolution and Optimum Seeking</u>. New York, John Wiley & Sons.

Sewell, W. and V. Shah (1968). "Social Class, Parental Encouragement, and Educational Aspirations." American Journal of Sociology **73**: 559-572.

Simon, H. A. (1986). Why Should Machines Learn. Machine Learning: An Artificial Intelligence Approach. R. Michalski, J. G. Carbonell and T. M. Mitchell. Los Altos, CA, Morgan Kaufmann Publishers. **1:** 25-37.

Singh, M. (1997). Learning Bayesian Networks from Incomplete Data. Proceedings of AAAI, Providence RI.

Spears, W. M. (1994). Simple Subpopulation Schemes. Proceedings of the Third Annual Conference on Evolutionary Programming, San Diego, World Scientific.

Spiegelhalter, D. J., A. P. Dawid, et al. (1993). "Bayesian Analysis in Expert Systems." Statistical Science **8**(3): 219-283.

Spirtes, P., C. Glymour, et al. (1993). Causation, Prediction, and Search. New York, Springer-Verlag.

Sutton, R. S. and A. G. Barto (1998). Reinforcement Learning. Cambridge, MA, The MIT Press.

Suzuki, J. (1993). A Markov Chain Analysis on a Genetic Algorithm. Proceedings of the 5th International Conference on Genetic Algorithms, Urbana-Champaign.

Syswerda, G. (1989). Uniform Crossover in Genetic Algorithms. Proceedings of the 3rd International Conference on Genetic Algorithms, Morgan Kaufman.

Turing, A. M. (1995). Computing Machines and Intelligence. Computation & Intelligence. G. F. Luger. Menlo Park, AAAI Press/The MIT Press: 23-46.

UAI98 (1998). . Proceedings of the Fourteenth Conference on Uncertainty in Artificial Intelligence, Madison, WI, Morgan Kaufmann.

Valiant, L. G. (1984). "A Theory of the Learnable." Communications of the ACM **27**(11): 1134-1142.

van Lint, J. H. and R. M. Wilson (1996). A Course in Combinatorics. Cambridge, Cambridge University Press.

Vapnik, V. (1995). The Nature of Statistical Learning Theory. New York, Springer.

Vapnik, V. N. (1998). Statistical Learning Theory. New York, John Wiley & Sons.

Vose, M. D. (1993). Modeling Simple Genetic Algorithms. <u>Foundations of Genetic Algorithms</u>. L. D. Whitley. San Mateo, CA, Morgan Kaufmann Publishers. **2:** 63-74.

Watson, S. R. and D. M. Buede (1987). <u>Decision Synthesis: The Principles and Practice of Decsion Analysis</u>. Cambridge, Cambridge University Press.

Weiss, G. M. (1995). <u>Learning with Rare Cases and Small Disjuncts</u>. The XII International Conference on Machine Learning, Tahoe City, CA, Morgan Kauffmann Publishers.

Whitley, D., J. R. Beveridge, et al. (1997). <u>Messy Genetic Algorithms for Subset Feature Selection</u>. Proceedings of the Seventh International Conference on Genetic Algorithms, East Lansing, MI, Morgan Kauffmann Publishers.

Whittaker, J. (1990). <u>Graphical Models in Applied Multivariate Statistics</u>. Chichester, John Wiley & Sons.

Winston, P. H. (1992). <u>Artificial Intelligence</u>. Reading, MA, Addison-Wesley Publishing Company.

Wolpert, D. H., Ed. (1995). <u>The Mathematics of Generalization</u>. Reading, Addison-Wesley Publishing Company.

CURRICULUM VITAE

James W. Myers was born on October 25, 1958, in Cullman Alabama and is an American citizen. He received a Bachelor of Science degree in Biology from the University of Montevallo in 1980, a Bachelor of Science in Electrical Engineering degree from Auburn University in 1983, a Master of Science degree in Computer Science from Troy State University in 1991, and a Doctor of Philosophy degree from George Mason University in the spring of 1999.

Dr. Myers is a Lieutenant Colonel in the United States Air Force and has served his country for over 18 years. Most of his career has been in acquisition and research and development for ballistic missile systems.